

**INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SÃO PAULO**

**DESENVOLVIMENTO DE MODELO MATEMÁTICO E SIMULAÇÃO DE UM ROBÔ  
MANIPULADOR DO TIPO SCARA**

**JOÃO VICTOR DE OLIVEIRA  
RHENAN DIAS MORAIS**

**GUARULHOS – SP  
2018**

**JOÃO VICTOR DE OLIVEIRA**

**RHENAN DIAS MORAIS**

**DESENVOLVIMENTO DE MODELO MATEMÁTICO E SIMULAÇÃO DE UM ROBÔ  
MANIPULADOR DO TIPO SCARA**

Trabalho de conclusão de curso orientado pelo Professor Me. Rogério Daniel Dantas, a ser apresentado no Instituto Federal de Ciência e Tecnologia de São Paulo, Campus Guarulhos como requisito básico para a conclusão do curso de Tecnologia em Automação Industrial.

**GUARULHOS – SP  
2018**

*“Pai se este cálice de sofrimento não pode ser afastado de mim sem que eu beba, então que seja feita a sua vontade.” - Mateus 26:39*

## RESUMO

O termo robô tem origem etimológica na palavra tcheca *robota*, que significa *trabalho forçado*, e no imaginário popular sua imagem se aproxima da forma humana, na qual existe um autômato capaz de realizar as mesmas tarefas e de forma similar aos humanos. Entretanto, o conceito de robô se atualiza constantemente, e desde o início do século XX, com a necessidade de melhoria na qualidade dos produtos e aumento da produtividade, a robótica industrial ganha corpo e encontra suas aplicações. Com o crescente avanço computacional em *hardware* e *software*, a robótica recebe suporte para grandes saltos no desenvolvimento tecnológico e integração com ambientes industriais, sendo possível obter velocidades de processamento superiores às das últimas décadas e maior possibilidade de previsão do funcionamento de um mecanismo robótico. Este projeto, objetiva desenvolver um modelo matemático para aplicação em um robô manipulador do tipo SCARA, simulando o funcionamento do mecanismo com a utilização da técnica *Hardware in the Loop* e o *software* de simulação de robôs *V-Rep*, em que a lógica de controle será embarcada em um ambiente controlado e suas respostas monitoradas virtualmente, sendo possível visualizar o funcionamento do manipulador por meio de um modelo tridimensional. Essa técnica permite ajustes durante o desenvolvimento do mecanismo robótico sem o risco de danificar uma instalação em ambiente real, com a característica de manter as respostas do sistema as mais próximas possíveis da aplicação final.

**Palavras chave:** Robótica industrial; Modelo matemático; Hardware in the Loop; Manipulador SCARA.

## ABSTRACT

The term robot has an etymological origin in the Czech word *robota*, which means forced labor, and in the popular imagination your image is close to the human form, in which there is an automaton capable of performing the same tasks and in a similar way to humans. However, the concept of robot is constantly updated, and since the beginning of the twentieth century, with the need to improve product quality and increase productivity, industrial robotics gains body and finds its applications. With the increasing computational advancement in hardware and software, robotics receives support for large steps in technological development and integration with industrial environments, being possible to obtain processing speeds superior to the last decades and greater possibility of prediction of the operation of a robotic mechanism. This project aims to develop a mathematical model for application in a robot manipulator of the SCARA type, simulating the operation of the mechanism using the Hardware in the Loop technique and the V-Rep robot simulation software, in which the control logic will be embedded in a controlled environment and the responses monitored virtually, being possible to visualize the operation of the manipulator by means of a three-dimensional model. This technique allows adjustments during the development of the robotic mechanism without the risk of damaging an installation in real environment, with the characteristic of keeping the system responses as close as possible to the final application.

**Keywords:** Industrial robotics; Mathematical model; Hardware in the loop; SCARA manipulator.

## LISTA DE ILUSTRAÇÕES

<b>Figura 1:</b> Estátuas no relógio de São Marcos (A) e no Old Town Hall Tower (B) (NIKU, 2013). ....	12
<b>Figura 2:</b> Esquema de configuração de um robô SCARA (CRAIG, 2012). ....	14
<b>Figura 3:</b> Esquema simplificado de um sistema HIL (Traduzido de National Instruments, 2016). ....	15
<b>Figura 4:</b> Obras de literatura sobre robôs da autoria de Isaac Asimov (FC Editora). ....	19
<b>Figura 5:</b> Gort (a) (20th Fox, 1951) e O Exterminador do Futuro (b) (Orion Pictures, 1987). ....	20
<b>Figura 6:</b> Sistema de transmissão (a) e elos e juntas em um robô (b) (PAZOS, 2011) ....	25
<b>Figura 7:</b> Atuador pneumático (a) (Festo, 2018), e atuador hidráulico (b) (HiComp, 2018). ....	26
<b>Figura 8:</b> Esquema simplificado de uma junta. (Adaptado de CRAIG, 2012). ....	27
<b>Figura 9:</b> Representação dos tipos de juntas existentes. (CRAIG, 2012). ....	28
<b>Figura 10:</b> Subclassificações de juntas existentes (GROOVER, 1998). ....	29
<b>Figura 11:</b> Worskpace cartesiano (a) (ROSÁRIO, 2004) e robô cartesiano (b) (Milacron, 2018). ....	30
<b>Figura 12:</b> Workspace cilíndrico (a) (ROSÁRIO, 2004) e robô cilíndrico (b) (ST Robotics, 2018). ....	31
<b>Figura 13:</b> Workspace polar (a) (ROSÁRIO, 2004) e robô Robot L-1000 (Fanuc, 2018). ....	32
<b>Figura 14:</b> Worskpace de revolução (a) (ROSÁRIO, 2004) e robô articulado (b) (KUKA, 2018). ....	32
<b>Figura 15:</b> Worskpace SCARA (a) (ROSÁRIO, 2004) e robô SCARA (b) (EPSON, 2018). ....	33
<b>Figura 16:</b> Modelo básico do robô SCARA (GROOVER, 1998). ....	35
<b>Figura 17:</b> Estrutura análoga de um sistema HIL (Traduzido de National Instruments, 2016). ....	38
<b>Figura 18:</b> Visão do software V-Rep, com alguns robôs disponíveis (Coppelia Robotics, 2018). ....	39
<b>Figura 19:</b> Diagrama estrutural da V-Rep API (Coppelia Robotics, 2018). ....	41
<b>Figura 20:</b> Placa Arduino Uno (Arduino, 2018). ....	42
<b>Figura 21:</b> Família de placas Arduino (Arduino, 2018). ....	43
<b>Figura 22:</b> Interface de um Robô industrial KUKA (KUKA, 2018). ....	44
<b>Figura 23:</b> Arquitetura geral do projeto. ....	48
<b>Figura 24:</b> Estrutura de um ciclo de comunicação de movimento. ....	49
<b>Figura 25:</b> Modelo Tridimensional do robô SCARA, no software V-Rep. ....	50
<b>Figura 26:</b> Modelo básico de um manipulador SCARA (GROOVER, 1998). ....	51
<b>Figura 27:</b> Robô SCARA em modelo trigonométrico (GROOVER, 1998). ....	54
<b>Figura 28:</b> Dualidade de acesso a uma posição no espaço (GROOVER, 1998). ....	56
<b>Figura 29:</b> Console IPython com resultados de saída do modelo programado. ....	63
<b>Figura 30:</b> Workspace do robô SCARA gerado com o modelo programado. ....	66
<b>Figura 31:</b> Teste de posicionamento com o modelo programado. ....	67
<b>Figura 32:</b> Teste de posicionamento com dois acessos a um ponto no espaço. ....	68
<b>Figura 33:</b> Modelo tridimensional SCARA e seus elementos. ....	69
<b>Figura 34:</b> Acionamento da junta prismática no modelo tridimensional. ....	70
<b>Figura 35:</b> Funcionamento da ferramenta de sucção do modelo tridimensional. ....	70
<b>Figura 36:</b> Visão geral do modelo tridimensional no V-Rep. ....	71
<b>Figura 37:</b> Árvore de elementos do modelo tridimensional no V-Rep. ....	72
<b>Figura 38:</b> Demonstração da ferramenta de construção de gráficos do V-Rep. ....	90
<b>Figura 39:</b> Comparação entre funções com e sem interpolação de juntas. ....	92

<b>Figura 40:</b> Gráficos das funções com interpolação ligada e desligada. ....	92
<b>Figura 41:</b> Resultado da rotina de simulação com função de movimento entre pontos. ....	93
<b>Figura 42:</b> Gráfico de resultado da rotina com função de movimento entre pontos. ....	94
<b>Figura 43:</b> Resultado da rotina de simulação com movimentação linear. ....	95
<b>Figura 44:</b> Gráfico de resultado da simulação com movimento linear. ....	96
<b>Figura 45:</b> Teste de simulação de escrita com função linear. ....	97
<b>Figura 46:</b> Gráfico de resultado da simulação de escrita com função linear. ....	97
<b>Figura 47:</b> Resultado da simulação em um plano tridimensional. ....	98
<b>Figura 48:</b> Gráfico de resultado da simulação em um plano tridimensional. ....	99
<b>Figura 49:</b> Teste de captura e posicionamento de objetos. ....	100
<b>Figura 50:</b> Captura e posicionamento sem ação de correção de velocidade angular. ....	101
<b>Figura 51:</b> Captura e posicionamento com ação de correção de velocidade angular. ....	101
<b>Figura 52:</b> Ambiente montado para simulação de posicionamento. ....	102
<b>Figura 53:</b> Simulação de rotina com processo de montagem. ....	103
<b>Figura 54:</b> Últimas etapas da rotina de simulação de montagem. ....	104
<b>Figura 55:</b> Simulação de montagem finalizada. ....	105
<b>Figura 56:</b> Montagem finalizada com maior nível de detalhes. ....	105

## LISTA DE TABELAS

<b>Tabela 1:</b> Alguns fatos históricos no desenvolvimento da robótica (GROOVER, 1998). .....	22
<b>Tabela 2:</b> Tabela de definição de comprimentos e ângulos (GROOVER, 1998). .....	35
<b>Tabela 3:</b> Possibilidades de simulação de um sistema HIL (Traduzido de Sarhadi, 2013). .....	38
<b>Tabela 4:</b> Tabela de aquisição de dados Denavit-Hartenberg. ....	52
<b>Tabela 5:</b> Tipos de dados int e float para a plataforma Arduino. ....	77
<b>Tabela 6:</b> Conversão de dados para o processo de transmissão. ....	78
<b>Tabela 7:</b> Conversão de dados para o processo de recepção. ....	79



## **LISTA DE ABREVIATURAS**

**SCARA:** *Selective Compliant Assembly Robot Arm*

**HIL:** *Hardware in the Loop*

**V-Rep:** *Virtual Robot Experimentation Platform*

**ISO:** *International Organization for Standardization*

**RIA:** *Robotic Industries Association*

**PUMA:** *Programmable Universal Machine for Assembly*

**API:** *Application Programing Interface*

## SUMÁRIO

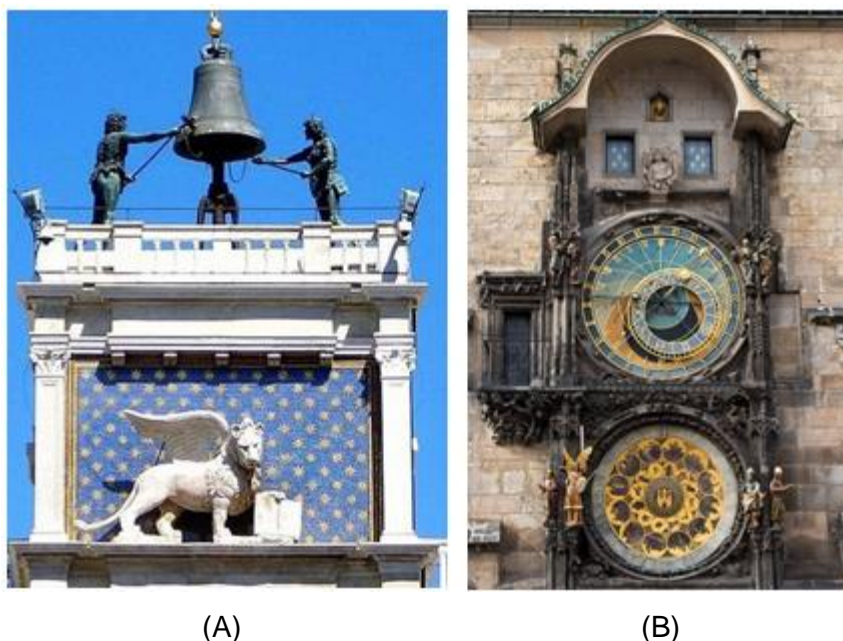
<b>1. INTRODUÇÃO .....</b>	<b>12</b>
1.1. MECANISMOS ROBÓTICOS .....	13
1.2. ROBÔ MANIPULADOR SCARA .....	14
1.3. HARDWARE IN THE LOOP.....	15
1.4. OBJETIVOS .....	16
1.4.1. OBJETIVO GERAL.....	16
1.4.2. OBJETIVOS ESPECÍFICOS .....	16
1.5. JUSTIFICATIVA .....	17
<b>2. FUNDAMENTAÇÃO TEÓRICA .....</b>	<b>18</b>
2.1. ORIGENS DA ROBÓTICA .....	18
2.2. ROBÓTICA NA FICÇÃO CIENTÍFICA .....	18
2.3. BREVE HISTÓRICO DA ROBÓTICA.....	21
2.4. CONCEITOS DE ROBÓTICA .....	24
2.4.1. DEFINIÇÃO DE ROBÔ.....	24
2.4.2. ELEMENTOS DE UM ROBÔ INDUSTRIAL .....	24
2.4.4. ATUADORES .....	26
2.4.5. JUNTAS.....	27
2.5. CLASSIFICAÇÃO DE ROBÔS.....	30
2.5.1. ROBÔ DE COORDENADAS CARTESIANAS .....	30
2.5.2. ROBÔ DE COORDENADAS CILÍNDRICAS.....	31
2.5.3. ROBÔ DE COORDENADAS POLARES (ESFÉRICAS).....	31
2.5.4. ROBÔ DE COORDENADAS DE REVOLUÇÃO (ARTICULADO).....	32
2.5.6. ROBÔ SCARA.....	33
2.6. CINEMÁTICA INVERSA E DIRETA (DENAVID-HARTENBERG) .....	34
2.7. CONCEITO DE HIL (HARDWARE IN THE LOOP) .....	37
2.8. SOFTWARE DE SIMULAÇÃO (V-REP).....	39
2.8.1. MOTORES DE SIMULAÇÃO.....	40
2.8.2. PROGRAMAÇÃO EM LUA.....	40
2.8.3. V-REP API.....	41
2.9. PLATAFORMA ARDUINO .....	42
2.10. PROGRAMAÇÃO DE ROBÔS.....	44
2.11. MÉTODOS DE PROGRAMAÇÃO .....	45
2.12. PROGRAMAÇÃO ON-LINE .....	46

2.12.1.	PROGRAMAÇÃO POR APRENDIZAGEM .....	46
2.12.2.	PROGRAMAÇÃO POR LINGUAGEM TEXTUAL .....	46
2.13.	PROGRAMAÇÃO OFF-LINE .....	46
<b>3.</b>	<b>MATERIAIS E MÉTODOS .....</b>	<b>47</b>
3.1.	ESTRUTURA DO PROJETO .....	47
3.2.	MODELAGEM MATEMÁTICA .....	51
3.2.1.	MODELAGEM MATRICIAL .....	51
3.2.2.	MODELAGEM TRIGONOMÉTRICA.....	54
3.3.	MODELADEM EM PYTHON.....	58
3.3.1	BIBLIOTECAS E DEPENDÊNCIAS.....	58
3.2.1.	CINEMÁTICA DIRETA EM PYTHON .....	59
3.3.2.	CINEMÁTICA INVERSA EM PYTHON.....	60
3.3.3.	TESTANDO O MODELO PROGRAMADO.....	61
3.2.4.	VISUALIZAÇÃO DA ÁREA DE TRABALHO.....	64
3.2.5.	TESTE DE POSICIONAMENTO .....	67
3.4.	MODELO TRIDIMENSIONAL .....	69
3.5.	MODELAGEM EM ARDUINO .....	73
3.4.1.	CINEMÁTICA DIRETA EM ARDUINO.....	74
3.4.2.	CINEMÁTICA INVERSA EM ARDUINO .....	75
3.6.	INTERFACES DE COMUNICAÇÃO .....	77
3.6.1.	INTERFACE ARDUINO – PYTHON .....	77
3.6.2.	INTERFACE PYTHON – V-REP.....	82
3.7.	FUNÇÕES DE MOVIMENTO.....	85
<b>4.</b>	<b>RESULTADOS E DISCUSSÃO .....</b>	<b>90</b>
4.1.	MOVIMENTO DE JUNTAS .....	91
4.2.	MOVIMENTO ENTRE PONTOS .....	93
4.3.	MOVIMENTO LINEAR .....	94
4.4.	MOVIMENTO NO EIXO Z .....	98
4.5.	CAPTURE E POSICIONAMENTO DE OBJETOS .....	100
4.6.	TESTE DE MONTAGEM.....	102
<b>5.</b>	<b>CONCLUSÃO.....</b>	<b>106</b>
	<b>REFERÊNCIAS BIBLIOGRÁFICAS .....</b>	<b>108</b>

## 1. INTRODUÇÃO

Automação e robótica são duas tecnologias que se desenvolveram de maneira diretamente relacionada. Segundo GROOVER (1988), em um contexto industrial podemos definir a automação como uma tecnologia que se ocupa do uso de sistemas mecânicos, eletrônicos e computadorizados na operação e controle da produção. Assim, a robótica é classificada como uma forma de automação industrial.

A ideia de robótica está presente nas mentes dos seres humanos desde muito cedo, a partir do momento em que o homem percebeu que era possível construir coisas. Artesões da idade média já criavam máquinas com o objetivo de imitar o movimento humano, um dos exemplos são as estátuas na torre do relógio de São Marcos em Veneza, que batem o sino na hora, e as estatuetas do século XV que contam uma história ao lado do *Old Town Hall Tower*, no Relógio Astronômico em Praga, como observado na Figura 1 abaixo.



**Figura 1:** Estátuas no relógio de São Marcos (A) e no Old Town Hall Tower (B) (NIKU, 2013).

Embora, no imaginário popular, o conceito de robô represente uma figura que se aproxima da forma humana, este conceito se atualiza constantemente, e as aplicações de robótica ganham cada vez mais força em ambientes industriais. É preciso então, definir o que classifica um mecanismo como robótico, ou pertencente a outro segmento, como por exemplo, um manipulador.

## **1.1. MECANISMOS ROBÓTICOS**

Não é possível intitular qualquer mecanismo automatizado como robótico. Tomando por exemplo a comparação entre um robô manipulador qualquer e um guindaste, é possível perceber uma diferença fundamental entre eles: o guindaste é controlado por um humano, enquanto o robô é pré-programado para executar uma tarefa. Apesar de exercerem a mesma função e serem bastante semelhantes, essa diferença determina se um dispositivo é um simples manipulador ou um robô.

Em geral, um robô é projetado e destinado a ser controlado por um computador, ou dispositivo semelhante, e tem seus movimentos redigidos a partir de técnicas que lidam com conceitos como graus de liberdade, cinemática, dinâmica e planejamento de trajetórias, e uma de suas notáveis vantagens incluem o fato de executarem movimentos precisos e eficientes, desde que projetados e modelados adequadamente.

Para NIKU (2013), robôs sozinhos quase nunca são úteis, eles são usados em conjunto com outros dispositivos, periféricos e outras máquinas de fabricação para executar uma tarefa ou fazer uma operação. O modelo de controle é parte integrante desse conjunto de dispositivos, devendo ser escolhido adequadamente com a estrutura do mecanismo a ser utilizado.

Quanto aos elementos e partes pertencentes a um robô, pode-se definir um mecanismo robótico como um conjunto de dispositivos que são integrados para formar um todo. Suas partes envolvem atuadores, responsáveis por executar os movimentos; sensores, responsáveis por coletar informações sobre o estado do robô; controlador, responsável por processar e controlar os movimentos; e atuador final ou extremidade, parte atuante do robô que irá conter a ferramenta ou elemento de manipulação coerente com o objetivo final da aplicação.

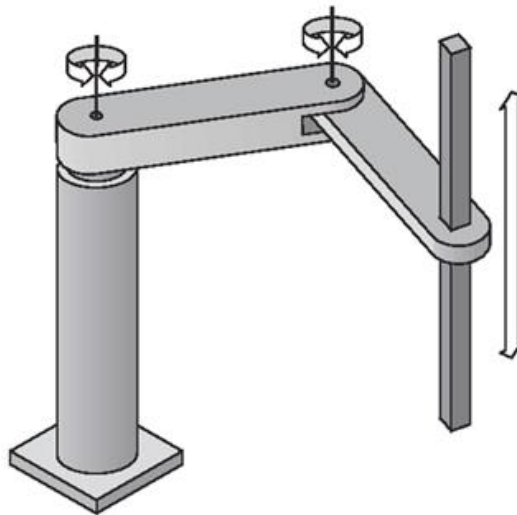
Tratando-se de modelamento de mecanismos robóticos, existem dois tipos de abordagem: a cinemática inversa, e cinemática direta. Como define CRAIG (2012), cinemática é a ciência que trata do movimento sem considerar as forças que o causam, são estudadas posição, velocidade e aceleração. As equações de cinemática direta podem determinar onde estará o terminal do robô caso todas as configurações de juntas sejam conhecidas, e a cinemática inversa permite calcular qual deve ser cada configuração a fim de localizar o robô em uma posição desejada.

## 1.2. ROBÔ MANIPULADOR SCARA

Uma das principais características que define um robô são os graus de liberdade que este possui, em outras palavras, a quantidade de movimentos individuais que o mecanismo pode fazer (NIKU, 2013).

A movimentação dos graus de liberdade se faz possível através das juntas que estão presentes no robô. Juntas são os elementos que ligam as partes rígidas do mecanismo robótico, chamadas de elos, e podem ter diferentes configurações de movimento, o que resulta em diferentes quantidades de pontos no espaço acessíveis ao manipulador (ROSÁRIO, 2004).

A configuração SCARA, acrônimo de *Selective Compliant Assembly Robot Arm* (braço robótico de montagem com complacência seletiva), apresenta duas (ou três) juntas rotacionais paralelas, permitindo movimento em um plano, e uma junta prismática perpendicular a este plano permitindo um movimento vertical para o atuador final. A Figura 2 mostra o esquema simplificado de configuração de um robô SCARA:



**Figura 2:** Esquema de configuração de um robô SCARA (CRAIG, 2012).

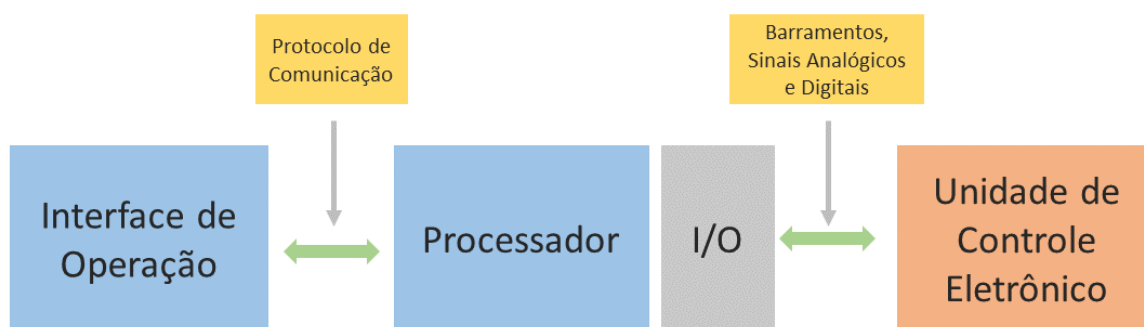
A aplicação de robôs SCARA é bastante comum em operações de montagem, uma vez que suas características são mais compatíveis com o plano x-y, mas com pouca flexibilidade ao longo do eixo z. Sua configuração e disposição de atuadores permite que o robô se movimente com muita rapidez, no entanto, com capacidade de carga útil reduzida.

Apesar de possuir área de trabalho menor do que alguns modelos de robôs articulados, apresenta diversas vantagens como velocidade, e relativa facilidade de programação do modelo cinemático, consequência de sua organização física de juntas.

### 1.3. HARDWARE IN THE LOOP

*Hardware in the Loop* (HIL), é um conceito que utiliza um sistema embarcado para simular o comportamento de um controlador em um ambiente real. É possível utilizar esta técnica para prever o comportamento de partes do sistema que apresentem algum desafio no desenvolvimento, buscando obter um comportamento o mais próximo possível da aplicação final, sem correr o risco de danificar partes atuantes do sistema durante testes de programação e desenvolvimento. (*National Instruments*, 2016).

Um sistema com HIL é composto basicamente por uma interface para operação, um processador que executa as funções em tempo real, e um conjunto de entradas e saídas. O processamento de tempo real é normalmente necessário para a simulação exata das partes do sistema que não estão fisicamente presentes no teste. O esquema de um sistema HIL simplificado por ser observado na Figura 3.



**Figura 3:** Esquema simplificado de um sistema HIL (Traduzido de National Instruments, 2016).

A interface do operador é utilizada para comunicação com o *hardware*, fornecendo visualização e monitoramento do sistema, além de possibilitar a execução de tarefas como gerenciamento da configuração, automação de testes, análise e gerenciamento de relatórios, entre outras (National Instruments, 2016).

Os sinais de entrada e saída podem ser quaisquer sinais analógicos, digitais ou de barramentos, que interagem com a unidade em teste, e são utilizados para produzir sinais de estímulo, adquirir dados de registro e análise e fornecer as interações com sensores e atuadores entre a unidade de controle eletrônico (ECU) e o ambiente virtual do operador.

#### **1.4. OBJETIVOS**

Os objetivos traçados com o desenvolvimento deste projeto apresentam-se da seguinte forma:

##### **1.4.1. OBJETIVO GERAL**

Desenvolver um modelo matemático para um robô manipulador do tipo SCARA, utilizando a técnica de *Hardware in the Loop* e softwares de simulação para monitorar o comportamento e as respostas obtidas com o modelo desenvolvido.

##### **1.4.2. OBJETIVOS ESPECÍFICOS**

- Definição das equações necessárias para criação de um modelo matemático que descreva de forma satisfatória o movimento do manipulador;
- Integração do modelo tridimensional com o software de simulação de robótica V-Rep (*Virtual Robot Experimentation Platform*);
- Desenvolver programação embarcada do modelo matemático utilizando técnica *hardware in the loop*;
- Integrar programação embarcada do modelo com a simulação do mecanismo robótico.



### **1.5. JUSTIFICATIVA**

Um mecanismo robótico com estrutura física bem projetada e modelo matemático bem desenvolvido oferece possibilidade de controle preciso e eficiente. O processo de desenvolvimento de um robô, no entanto, não é algo simples e por vezes pode acarretar na danificação da estrutura robótica, ou do produto manipulado. Para evitar tais riscos, são utilizadas técnicas de simulação em ambientes controlados, sendo possível prever o comportamento das atividades em um ambiente real.

Deste modo, o projeto justifica-se em aplicar e disseminar o uso da técnica de simulação embarcada para um modelo de robô manipulador SCARA, utilizando o software V-Rep e a plataforma de desenvolvimento *Arduino*, ambas de fácil acesso e com ampla documentação disponível para o usuário, desenvolvendo durante o processo, ferramentas que podem ser adaptadas e facilmente utilizadas em outros tipos de robôs manipuladores, ou móveis.

## **2. FUNDAMENTAÇÃO TEÓRICA**

### **2.1. ORIGENS DA ROBÓTICA**

O âmbito da robótica tem origem na ficção científica, com primeiros relatos por volta de 1920. Um dos trabalhos mais relevantes para a discussão das origens da robótica é a novela escrita por Mary Shelley, em 1817, intitulada *Frankenstein*, que relata os esforços do Dr. Frankenstein em criar um monstro humanoide, fato que aterroriza a comunidade local, como explica GROOVER (1998).

O termo em inglês *robot* deriva da palavra tcheca *robota*, cunhada em uma peça de autoria de Karel Capek, intitulada *Os Robôs Universais de Rossum*, e significa servidão, ou trabalho forçado. Na história, *Rossum*, um cientista brilhante, começa a fabricar robôs que possuem a função de servir a humanidade obedientemente e fazer todo o trabalho físico necessário. O plano toma um rumo não desejado quando os seres quase perfeitos desenvolvidos pelo cientista começam a não gostar dos seus papéis de serventes e rebelam-se contra seus senhores, destruindo toda a vida humana.

Para Niku (2013), apesar das limitações dos mecanismos robóticos conhecidos atualmente, o conceito popular de robô é de que ele age como o ser humano, imagem que foi popularizada pela ficção científica, que também criou a ideia de superioridade dos robôs sobre seus criadores humanos.

### **2.2. ROBÓTICA NA FICÇÃO CIENTÍFICA**

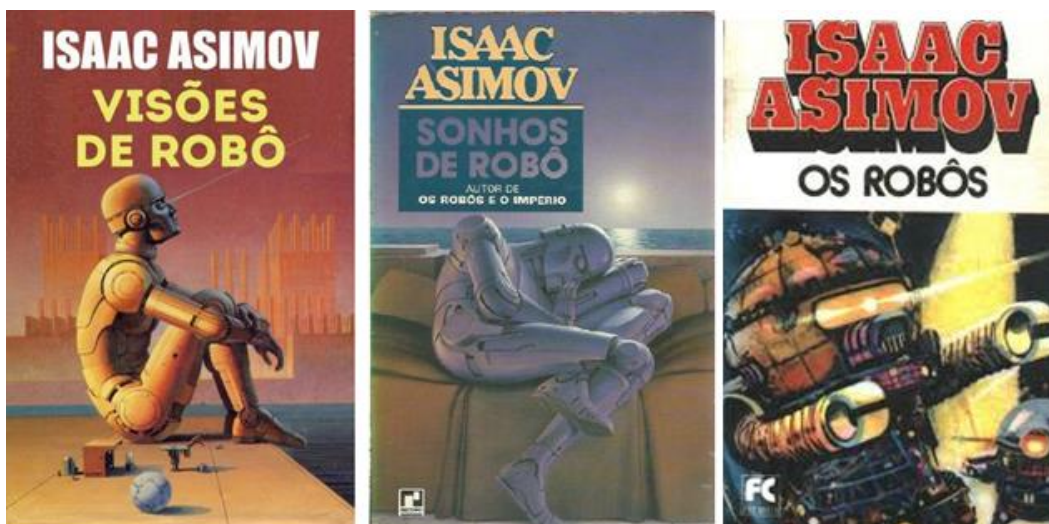
Temos por conceito na ficção aqueles robôs totalmente independentes, que executam funções sozinhos, autônomas, altamente inteligentes, temos robôs que se comunicam com seus mestres, ou até mesmo se voltam contra eles.

Os robôs tomaram um espaço gigante na ficção, sendo temas de livros filmes, desenhos animados até mesmo seriados de TV, tornando-se presente no mundo que hoje conhecemos, fazendo-nos imaginar como seria se os robôs fossem mesmo assim quase humanos (ROSÁRIO, 2004).

Alguns exemplos são peças como as obras de Isaac Asimov, alguns filmes como *Star Wars*, *O homem bicentenário*, *AI – Inteligência Artificial*, e nos desenhos com *Os Jetsons*, *Mega XLR*, *Robôs*, etc.

Isaac Asimov, entre os escritores de ficção científica, contribuiu com inúmeras histórias sobre robôs, e possui o crédito de ter desenvolvido o termo *robótica* em uma de suas obras. Em seu trabalho, a figura de robô é a de uma máquina bem projetada, livre de falhas e que atua de acordo com três princípios, chamados de *Três leis da Robótica*:

1. Um robô não pode ferir um ser humano ou, por inação, permitir que um humano seja ferido.
2. Um robô deve obedecer às ordens dadas por humanos, exceto quando isto conflitar com a Primeira Lei.
3. Um robô deve proteger sua própria existência, a menos que isto conflite com a Primeira ou segunda Lei.



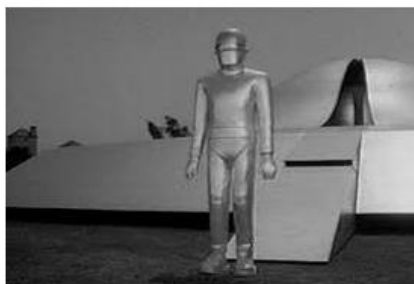
**Figura 4:** Obras de literatura sobre robôs da autoria de Isaac Asimov (FC Editora).

O conhecimento da robótica deve sua disseminação em grande parte a filmes e programas de televisão. Surge uma linha que se torna praticamente padrão para filmes de ficção: máquinas que foram feitas para ajudar as pessoas se rebelam e passam a lutar contra os seus criadores. Esse modelo de ficção científica é chamado por Asimov de “Complexo de Frankenstein”, como define GROOVER (1998).

O filme *O Dia em que a Terra Parrou*, de 1951, é considerada uma das primeiras obras cinematográficas com a presença de robôs. Mostrava uma missão vinda de um planeta distante, enviada a Terra em um disco voador tripulado por um robô onisciente, onipotente e indestrutível, nomeado *Gort*, apresentado na figura 5.

Já em *2001: Uma Odisseia no Espaço*, de 1969, a figura principal não era um robô mecânico, mas um computador inteligente, falante e com personalidade, chamado *HAL*. A tarefa do computador era controlar e monitorar os sistemas de uma espaçonave com destino ao planeta Júpiter.

Já em 1987, houve a estreia de um dos clássicos do cinema o *Exterminador do Futuro*, um Androide com partes humanas que viaja para o passado, exterminando outros androides do mal.



(a)



(b)

**Figura 5:** Gort (a) (20th Fox, 1951) e O Exterminador do Futuro (b) (Orion Pictures, 1987).

### **2.3. BREVE HISTÓRICO DA ROBÓTICA**

NIKU (2013) define que a criação ficcional dos robôs, seu deu com a fantasia de desenvolver uma máquina, análoga ao ser humano, capaz de pensar e agir de maneira inteligente, ainda que artificial, denotando uma vontade suprimida de ter uma espécie de “escravo metálico” que satisfaça todas suas vontades.

Apesar dos relatos antigos de criação de mecanismos robóticos, a ideia de construir robôs como são conhecidos hoje começou a tomar força no início do século XX, com a necessidade de aumentar a produtividade no meio industrial e gerar melhoria na qualidade dos produtos.

Como apresenta ROSÁRIO (2005), devido aos inúmeros recursos que os sistemas micro controlados nos oferecem, a robótica atravessa uma época de contínuo crescimento, que permitirá, em curto espaço de tempo, o desenvolvimento de robôs cada vez mais inteligentes.

ROSÁRIO (2005) reporta que, há algumas décadas atrás, os robôs faziam parte apenas do imaginário humano e da ficção científica, e no início dos anos 60, começaram a ser utilizados para substituir o homem em tarefas que ele não podia realizar, as quais envolviam condições desagradáveis como alto nível de calor, ruídos, gases tóxicos, radioatividade, esforço físico extremo, entre outras.

Nos últimos anos, a evolução dos robôs tem sido garantida através de duas tendências: o aumento do nível salarial dos empregados e o notável avanço tecnológico, que provoca uma redução no custo de desenvolvimento de robôs e uma melhoria significativa em seu desempenho. A Tabela 1 a seguir, apresenta uma linha do tempo com alguns fatos históricos considerados marcantes para o desenvolvimento da robótica.

**Tabela 1:** Alguns fatos históricos no desenvolvimento da robótica (GROOVER, 1998).

Período	Fato histórico
<b>Século IV A.C (Grécia)</b>	Aristóteles relata os primeiros princípios similares aos conhecidos atualmente como robótica, utilizando instrumentos com o intuito de auxiliar determinados trabalhos.
<b>Século XVIII</b>	Com a Revolução Industrial, surgem novos mecanismos e instrumentos que tornam possível a evolução do maquinário e realização de uma série de ações sequenciadas.
<b>Século XIX</b>	Introdução do motor elétrico na industrial, máquinas começam a substituir tarefas manuais.
<b>1914 – 1918</b>	Primeira Guerra Mundial, o poder da máquina mostra sua forma negativa e destrutiva.
<b>1921</b>	A palavra <i>robô</i> é utilizada pela primeira vez na ficção. O texto relata a criação de robôs para substituir o homem nos trabalhos pesados. O robô começa a ser visto como uma máquina humana, com inteligência e personalidade,
<b>1940</b>	Isaac Asimov desenvolve e publica as <i>Três Leis da Robótica</i> .
<b>1946</b>	O inventor norte-americano G. C. Devol desenvolve um dispositivo controlador capaz de registrar sinais elétricos magneticamente e reproduzi-los para operar uma máquina mecânica.
<b>1951</b>	Operadores por controle remoto são utilizados pela primeira vez para manipulação de materiais radioativos, demonstrando a possibilidade de mecanismos robóticos em substituir o homem em tarefas perigosas.
<b>1959</b>	Devol e Joseph F. Engerlberger desenvolvem primeiro robô industrial pela <i>Unimation Inc.</i> , capaz de executar automaticamente uma variedade de tarefas, além da possibilidade de ser reprogramado e remodelado para outras tarefas.
<b>1961</b>	O robô <i>Unimate</i> , de Devol, é instalado na <i>Ford Motor Company</i> para atender a uma máquina de fundição sob pressão.
<b>1969</b>	O homem pisa no solo lunar pela primeira vez. Já se utilizavam manipuladores robóticos para coleta de amostras e pequenas tarefas com comando por controle remoto.
<b>1974</b>	A <i>Cincinnati Milacron</i> apresenta o primeiro robô industrial controlado por computador, denominado <i>The Tomorrow Tool</i> , ou <i>T3</i> , era capaz de mover objetos em uma linha de montagem. O estudo sobre utilização de sensores ganha força.

<b>1974</b>	A <i>Kawasaki</i> instala robôs <i>Unimation</i> em operações de soldagem de arco para chassis de motocicleta.
<b>1975</b>	A <i>Olivetti</i> começa a utilizar o <i>Robô Sigma</i> em operações de montagem, uma das primeiras aplicações de montagem efetivas da robótica.
<b>1975</b>	Robô para montagem mecânica de uma máquina de escrever é desenvolvido na <i>IBM</i> por <i>Will e Grossman</i> .
<b>1978</b>	Robô <i>PUMA</i> ( <i>Programmable Universal Machine for Assembly</i> - Máquina Universal Programável para Montagem) é desenvolvido pela <i>Unimation</i> , utilizando como base projetos de estudos do <i>General Motors</i> .
<b>1979</b>	Desenvolvimento do robô do tipo <i>SCARA</i> na <i>Universidade Yamanashi</i> , no Japão. Vários robôs <i>SCARA</i> são introduzidos comercialmente por volta de 1981.
<b>1980</b>	A <i>General Motors</i> , em Detroit, nos Estados Unidos, introduz um robô industrial com capacidade de reconhecer diferentes componentes em uma linha de transporte e escolher aqueles que necessita.
<b>1986</b>	Produtos educacionais baseados em <i>LEGO</i> são colocados no mercado. <i>Honda</i> lança um projeto para construir um robô humanoide que possa andar.
<b>1997</b>	O primeiro torneio <i>Robocup</i> é realizado no Japão, o objetivo é ter uma equipe totalmente automatizada de robôs para vencer o mundo a melhor equipe de futebol no ano de 2050.
<b>1999</b>	É lançada a primeira versão do cão robótico <i>Aibo</i> , da <i>Sony</i> , o cão robótico com a capacidade de aprender e se comunicar com seu dono.
<b>2000</b>	É lançado pela <i>Honda</i> , o <i>ASIMO</i> , primeiro da nova linha de robôs humanoides da empresa.
<b>2002</b>	É introduzido no mercado o robô <i>Roomba</i> , um aspirador robótico que rapidamente ganha sucesso e demonstra força da aplicação doméstica da robótica.
<b>2011</b>	Diversas empresas investem em estudos para a fabricação de exoesqueletos que podem ser integrados com um corpo humano a fim de ajudar pessoas com paralisia.
<b>2012</b>	<i>Universidade de Pittsburgh</i> desenvolve braço mecânico que pode ser controlado pelo pensamento

## 2.4. CONCEITOS DE ROBÓTICA

### 2.4.1. DEFINIÇÃO DE ROBÔ

Conforme especificado pela *Robotic Industries Association* (RIA), um robô industrial é definido como um "manipulador multifuncional reprogramável projetado para movimentar materiais, partes, ferramentas ou peças especiais, através de diversos movimentos programados, para o desempenho de uma variedade de tarefas" (RIVIN, 1988).

Já a norma ISO (*International Organization for Standardization*) 10218 apresenta uma definição mais completa e define um robô como sendo: "uma máquina manipuladora com vários graus de liberdade controlada automaticamente, reprogramável, multifuncional, que pode ter base fixa ou móvel para utilização em aplicações de automação industrial".

### 2.4.2. ELEMENTOS DE UM ROBÔ INDUSTRIAL

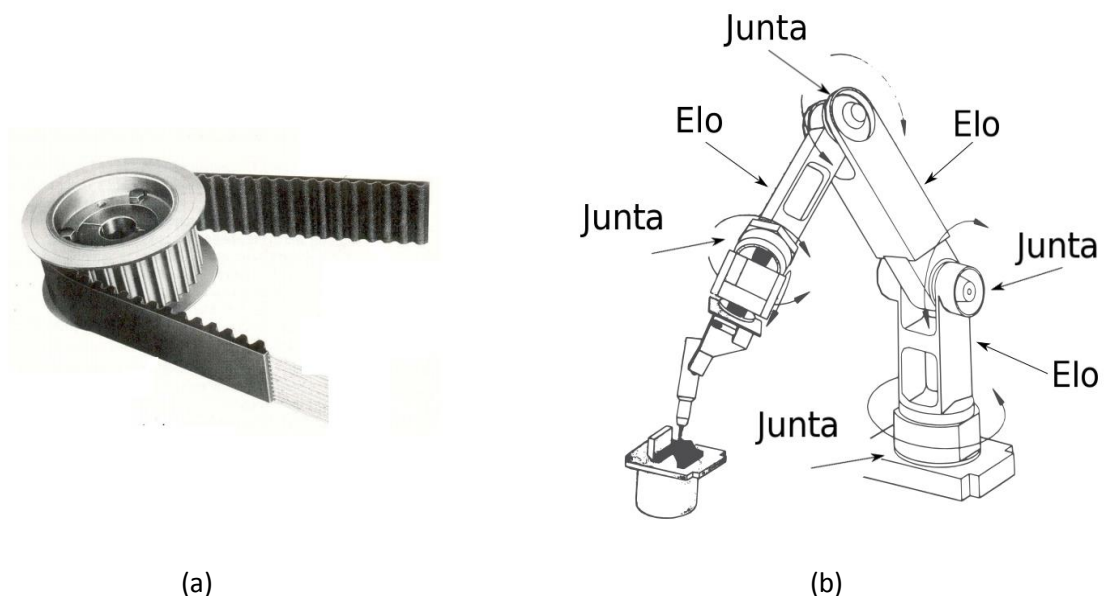
Segundo ROSÁRIO (2004), um robô industrial, pode ser descrito como a integração de alguns elementos presentes em sua estrutura, sendo eles:

- **Manipulador mecânico:** refere-se ao aspecto mecânico e estrutural do robô e a combinação de elementos estruturais rígidos (corpos ou elos) conectados entre si através de articulações (juntas).
- **Atuadores:** componentes que convertem energia elétrica, hidráulica ou pneumática, em potência mecânica.
- **Sensores:** são os elementos que fornecem parâmetros sobre o comportamento do manipulador, em termos de posição e velocidade, em outras palavras, servem para localizar o robô no espaço.
- **Unidade de controle:** responsável pelo gerenciamento e monitoração dos parâmetros responsáveis pela realização de tarefas do robô. De modo simples, é o cérebro do robô, que realiza interface entre todos os elementos, realizando a rotina desejada.
- **Efetuator:** É o elemento de ligação entre o robô e o meio que o cerca. Pode ser do tipo garra ou ferramenta. A ferramenta tem como função realizar uma ação ou trabalho sobre uma peça, sem necessariamente manipulá-la.



Quanto ao que diz respeito ao elemento manipulador mecânico citado na enumeração anterior, se destacam alguns conceitos importantes que devem ser observados: (ROSÁRIO, 2004).

- **Elos:** São os elementos rígidos dos robôs, são conectados entre si através de articulações, projetados para apresentar elevada rigidez aos esforços de flexão e torção.
- **Juntas:** São as articulações do robô, os elementos que unem os diferentes elos e é responsável pelo movimento do mecanismo. De modo geral, é possível dizer que o número de juntas equivale ao número de graus de liberdade.
- **Sistema de transmissão:** Os sistemas de transmissão são responsáveis por transmitir a potência mecânica dos atuadores aos elos. Por elementos de um sistema de transmissão, entendem-se engrenagens, correntes, correias dentadas, polias, entre outros.



**Figura 6:** Sistema de transmissão (a) e elos e juntas em um robô (b) (PAZOS, 2011)

#### 2.4.4. ATUADORES

Em relação aos atuadores, é possível utilizar diferentes tipos, como pneumáticos, hidráulicos ou elétricos. Segundo PAZOS (2011), de modo geral, é possível definir que atuadores pneumáticos e hidráulicos proporcionam movimentos lineares, enquanto motores elétricos proporcionam movimentos angulares.

- **Atuadores pneumáticos:** Os atuadores pneumáticos, são aqueles que como o nome sugere, atuam sob efeito pneumático, ou seja, tem seu funcionamento baseado na utilização de ar comprimido. (Figura 7).
- **Atuadores hidráulicos:** Já os atuadores hidráulicos, são aqueles que atuam sob a utilização de um fluido, um tipo de óleo específico para a aplicação hidráulica. (Figura 7).
- **Atuadores eletromagnéticos:** São os atuadores que tem seu funcionamento baseado em efeitos eletromagnéticos, em outras palavras, motores. São os tipos de atuadores mais utilizados na construção de mecanismos robóticos, principalmente os motores de corrente contínua e motores de passo.



(a)

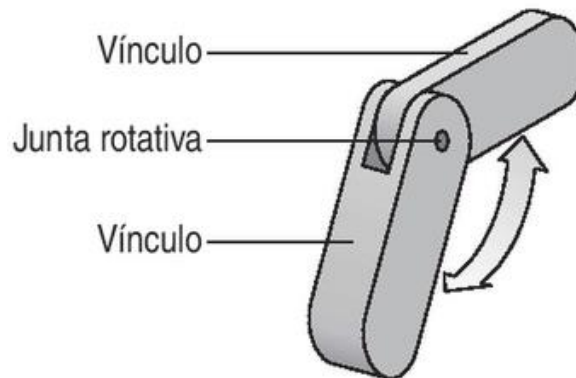


(b)

**Figura 7:** Atuador pneumático (a) (Festo, 2018), e atuador hidráulico (b) (HiComp, 2018).

#### 2.4.5. JUNTAS

Juntas são os elementos que unem os elos do robô e proporcionam que o mecanismo robótico realize movimentos. A Figura 6 apresenta um esquemático entre dois elos (vínculos) e uma junta do tipo rotacional. (CRAIG, 2012).

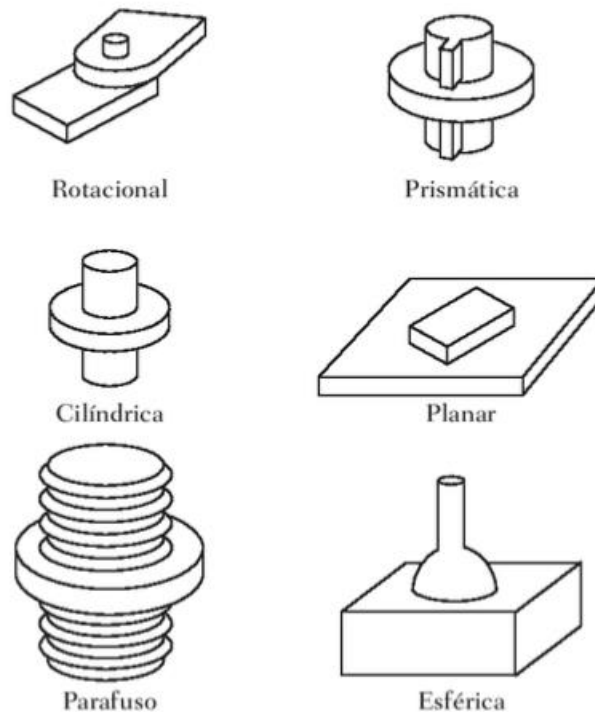


**Figura 8:** Esquema simplificado de uma junta. (Adaptado de CRAIG, 2012).

As juntas podem ser classificadas da seguinte maneira:

- **Rotacional:** São as juntas que giram em torno de um eixo de rotação, movimentam os elos através de uma dobradiça comum as duas partes, comumente representada pela atuação de um motor elétrico;
- **Prismática, ou linear:** É o tipo de junta que proporciona movimento linear, em linha reta, geralmente representada por um pistão, ou cilindro;
- **Cilíndrica:** É composta por uma união de duas juntas, uma rotacional e uma prismática, combinando os movimentos em torno de um eixo com o movimento linear;
- **Esférica:** Esta junta funciona com a combinação de três juntas rotacionais, combinando o movimento de três eixos de rotação em uma única junta;
- **Planar:** Composta por duas juntas prismáticas, realiza movimentos em duas direções, comumente usada em configurações cartesianas;
- **Parafuso:** Junta constituída por um parafuso com uma porca, ao qual é possível realizar movimento linear análogo ao da junta prismática, porém adaptando o movimento com fundamento rotacional em um eixo central (movimento do parafuso).

A Figura 9 abaixo apresenta as ilustrações esquemáticas de cada um dos tipos de juntas existentes e utilizadas em mecanismos robóticos.

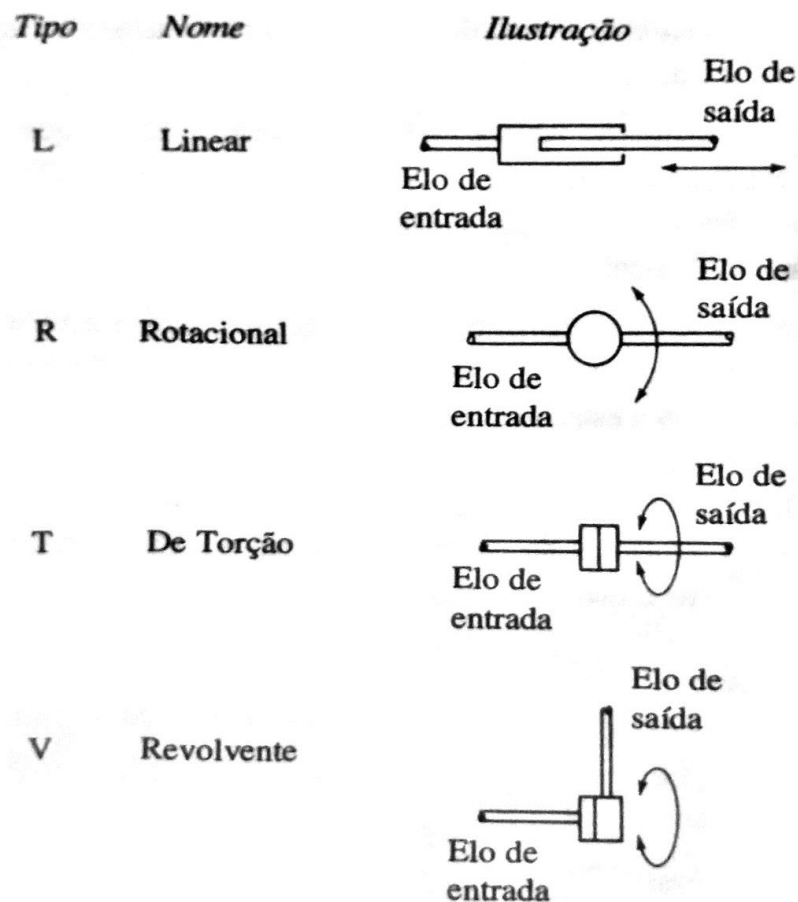


**Figura 9:** Representação dos tipos de juntas existentes. (CRAIG, 2012).

Segundo GROOVER (1998), no geral, robôs industriais utilizam apenas juntas prismáticas e rotativas e a direção dos elos de entrada e de saída em relação ao eixo de rotação classificam as juntas rotacionais, assim, tem-se as seguintes subclassificações de juntas rotativas:

- **Linear, L:** Na junta tipo L, o elo de saída mantém a direção do elo de entrada, realizando movimento linear;
- **Rotacional, R:** Na junta tipo R, o eixo de rotação é perpendicular aos eixos dos dois elos de conexão, realizando movimento rotacional;
- **Torção, T:** O junto tipo T é uma variação da junta rotacional, onde mesmo com movimento rotacional o movimento no elo de saída mantém a direção do elo de entrada;
- **Revolvente, V:** a junta do tipo V é uma terceira variação da junta R, onde o elo de entrada é paralelo ao eixo de rotação, e perpendicular ao elo de saída.

A ilustração a seguir, na Figura 10 (GROOVER, 1988) apresenta a esquematização teórica de cada um dos tipos de juntas existentes. Os diferentes tipos de juntas são combinados a fim de construir mecanismos robóticos e classificá-los conforme o tipo de movimento que este é capaz de realizar.



**Figura 10:** Subclassificações de juntas existentes (GROOVER, 1998).

Robôs industriais com frequência têm sua nomenclatura definida conforme a quantidade e tipo de juntas utilizadas. Por exemplo, um esquema simples de um robô do tipo SCARA, como apresentado na Figura 2, poderia ser classificado como um robô RRP.

## 2.5. CLASSIFICAÇÃO DE ROBÔS

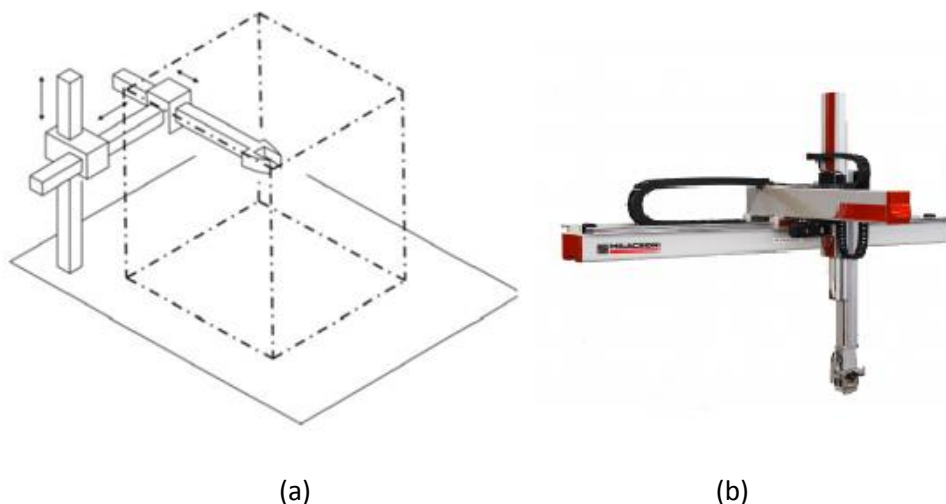
Os robôs industriais podem ser classificados de acordo com o número de juntas, o tipo de controle, o tipo de acionamento e a geometria. De fato, é usual classificar os robôs em relação ao seu espaço de trabalho (*workspace*), ou também em relação a estrutura de das juntas utilizadas.

Segundo ROSÁRIO (2004), existem cinco classes principais de manipuladores, segundo o tipo de junta, o que permite diferentes possibilidades de posicionamento no volume de trabalho.

### 2.5.1. ROBÔ DE COORDENADAS CARTESIANAS

É classificado como robô de coordenada cartesiana, aquele que pode se movimentar em linha reta, com deslocamentos horizontais e verticais. Recebe esse nome em função do modo a localizar um ponto no espaço, através de coordenadas em um plano cartesiano (x, y e z). (ROSÁRIO, 2004).

Possuem geralmente três juntas deslizantes, recebendo a classificação de um robô do tipo PPP (três juntas prismáticas), e são caracterizados pela grande exatidão na localização da extremidade. A Figura 11 ilustra o robô do tipo cartesiano.

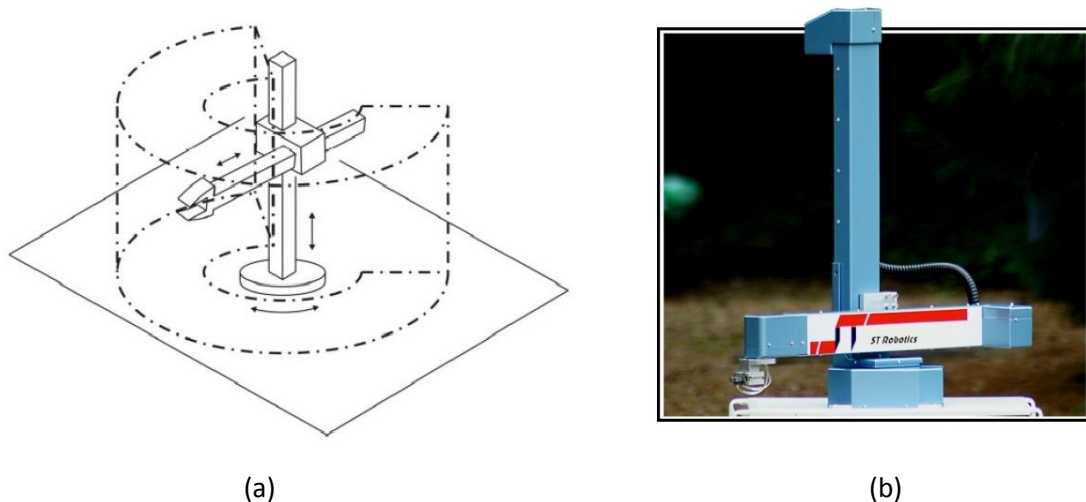


**Figura 11:** Worskspace cartesiano (a) (ROSÁRIO, 2004) e robô cartesiano (b) (Milacron, 2018).

### 2.5.2. ROBÔ DE COORDENADAS CILÍNDRICAS

Robôs deste tipo combinam a ação de dois tipos de juntas diferentes, lineares e rotacionais, gerando uma área útil de trabalho em um formato cilíndrico. É codificado como RPP, ou seja, consistem em uma junta de revolução combinada com duas juntas prismáticas. (ROSÁRIO, 2004).

Apesar de apresentar área de trabalho ligeiramente maior que a dos robôs cartesianos, apresenta maior complexidade no controle do posicionamento, uma vez que proporciona alcance ao mesmo ponto no espaço com posicionamentos diferentes. O robô do tipo cilíndrico pode ser observado na Figura 12 a seguir.

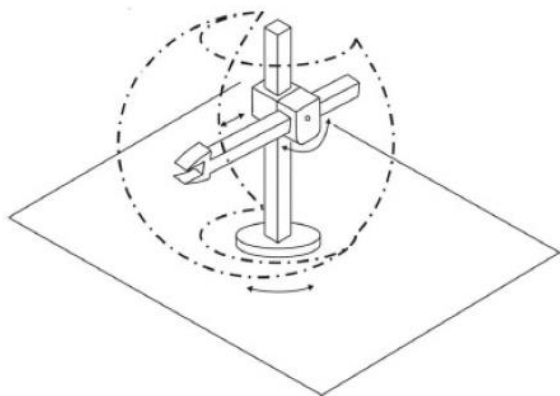


**Figura 12:** Workspace cilíndrico (a) (ROSÁRIO, 2004) e robô cilíndrico (b) (ST Robotics, 2018).

### 2.5.3. ROBÔ DE COORDENADAS POLARES (ESFÉRICAS)

Robôs de coordenadas esféricas são aqueles com três eixos que descrevem uma área de trabalho esférica. É composto por dois movimentos rotacionais, e um terceiro movimento linear, descrito assim como RRP.

Possui área de trabalho maior que os modelos cartesianos e cilíndricos, porém seu controle de posicionamento é ainda mais complexo, devido à sua área de trabalho em formato de esfera. Pode ser observado na Figura 13 um esquema do robô de coordenadas esféricas.



(a)

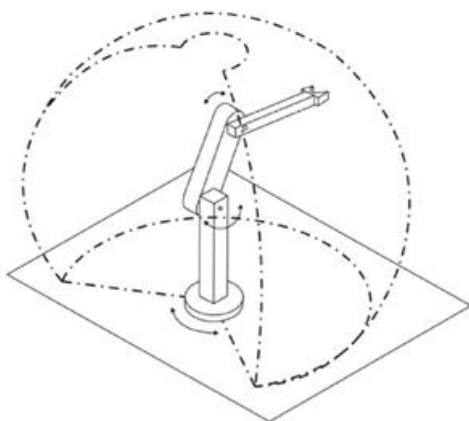


(b)

**Figura 13:** Workspace polar (a) (ROSÁRIO, 2004) e robô Robot L-1000 (Fanuc, 2018).

#### 2.5.4. ROBÔ DE COORDENADAS DE REVOLUÇÃO (ARTICULADO)

Os robôs de coordenadas de revolução, também chamados de articulados, se caracterizam por possuir três juntas de revolução, ou seja, são codificados como RRR. Foi inicialmente projetado para atender as necessidades na indústria automobilística, e um dos primeiros robôs comerciais desse tipo foi o PUMA (*Programmable Universal Machine for Assembly*). A Figura 14 mostra o esquema da área de trabalho de um robô de coordenadas de revolução.



(a)



(b)

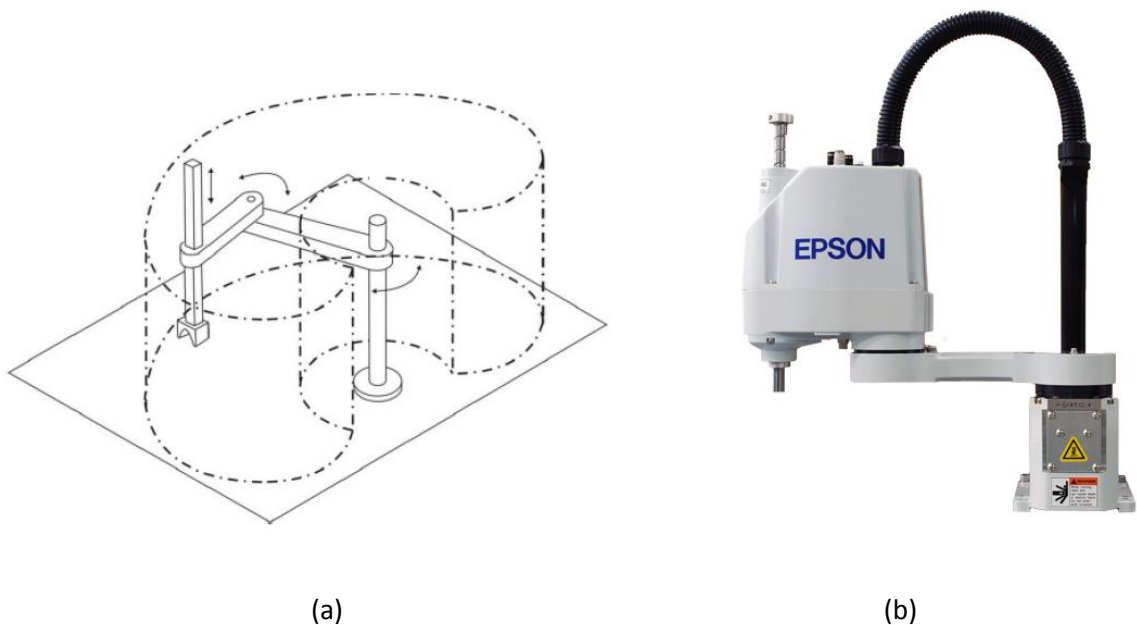
**Figura 14:** Worspakece de revolução (a) (ROSÁRIO, 2004) e robô articulado (b) (KUKA, 2018).



### 2.5.6. ROBÔ SCARA

O robô SCARA, é uma configuração mais recente em relação as outras utilizadas, é codificado como RRP, com duas juntas de revolução se movimentando no plano cartesiano, e uma junta prismática se movimentando no eixo z.

É utilizado principalmente em tarefas de montagem, possuindo uma área de trabalho limitada, porém grande velocidade de movimentação e precisão de localização. Pode ser confundido com um robô de coordenadas esféricas, devido a sua configuração RRP, porém apresenta área de trabalho e localização diferenciada. Na Figura 15 é possível observar a área de trabalho de um robô SCARA.



**Figura 15:** Worskpace SCARA (a) (ROSÁRIO, 2004) e robô SCARA (b) (EPSON, 2018).

## 2.6. CINEMÁTICA INVERSA E DIRETA (DENAVID-HARTENBERG)

A fim de desenvolver um esquema para controlar o movimento de um manipulador, é necessário desenvolver técnicas para demonstrar a posição do braço em pontos no tempo (GROOVER, 1998).

As juntas e os elos recebem letras e números para suas identificações, sendo as juntas rotuladas de  $Jn$ , onde  $n$  começa com 1 sendo a base do manipulador, e os elos  $Ln$ , esse também começado em 1, sendo este o elo que estiver mais próximo da base. A notação  $Ln$ , também é utilizada para indicar o comprimento de cada elo gerando assim derivações das equações.

Para um robô do tipo SCARA, configurado como RR, o modo mais usual é utilizar os ângulos das duas juntas  $\theta_1$  e  $\theta_2$ . Isto é conhecido como representação no espaço da “junta” e podemos defini-la como:

$$P_j = (\theta_1, \theta_2) \quad (1)$$

O outro modo de definir posição do braço é no espaço cartesiano. A origem é determinada pela base do robô. A posição do fim do braço seria definida no espaço cartesiano como:

$$P_w = (x, y) \quad (2)$$

Para comunicar o robô a outras máquinas que não tem uma compreensão detalhada da cinemática do robô em si, uma representação “neutra”, tal como o espaço cartesiano, tem de ser utilizada. (GROOVER, 1998).

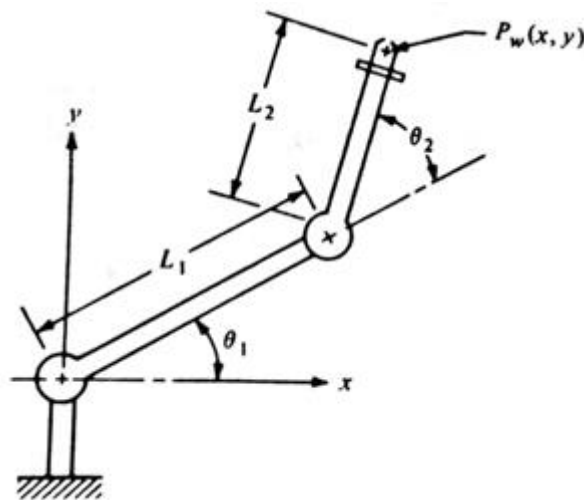
Para usarmos ambas as representações, devemos poder transformar de uma para outra. Ir do espaço de junta para o espaço cartesiano é chamado transformação direta e o mesmo acontece para o inverso, ir do plano cartesiano para espaço de junta, isso se chama “transformação inversa”.

Como define CRAIG (2012), cinemática é o método utilizado para descrever matematicamente a movimentação de diversos tipos de mecanismos robóticos, com este método é possível controlar diferentes tipos de robôs através dos cálculos cinemáticos utilizando cinemática inversa ou direta.

Para manipuladores como o SCARA, na cinemática direta são conhecidas as posições das articulações e obtêm-se a posição da extremidade do manipulador, já na cinemática inversa acontece o contrário da direta, dada a posição da extremidade, obtêm-se a posições correspondente das articulações (NIKU, 2013).

O método utiliza o conceito básico de matrizes de referência para encontrar o ponto desejado, aplicando cálculos com matrizes de referência (padrão), a fim de encontrar transformações de coordenadas, no caso do projeto desenvolvido, as coordenadas são descritas em um plano cartesiano, utilizando o conceito de *Denavit e Hartenberg*, que determina o movimento dos elos.

Seguindo o conceito físico do nosso projeto que é um robô de duas articulações de revolução (2R), tem-se uma modelo base conforme observado na Figura 16 abaixo.



**Figura 16:** Modelo básico do robô SCARA (GROOVER, 1998).

A Tabela 2 a seguir, apresenta a relação de elos e ângulos de movimento do modelo para um robô SCARA, seguindo os parâmetros de Denavit-Hartenberg.

**Tabela 2:** Tabela de definição de comprimentos e ângulos (GROOVER, 1998).

<i>Ligamento</i>	$a_i$	$\alpha_i$	$d_i$	$\theta_i$
1	a1	0	0	$\theta_1$
2	a2	0	0	$\theta_2$

O eixo Z do robô SCARA, não é uma das variáveis de cálculo, pois tem determinação independentemente do restante do robô, sendo possível implementar ferramentas, tanto pneumáticas, magnéticas, garras, de vácuo, entre outras.

As matrizes de transformação homogênea que determinam a posição final do robô são apresentadas nas equações 3 e 4, e a matriz de referência é observada na equação 5, obtidas de GROOVER (1998).

$$A = \begin{pmatrix} \cos \theta_1 & -\sin \theta_1 & 0 & a_1 \cos \theta_1 \\ \sin \theta_1 & \cos \theta_1 & 0 & a_1 \sin \theta_1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (3)$$

$$B = \begin{pmatrix} \cos \theta_2 & -\sin \theta_2 & 0 & a_2 \cos \theta_2 \\ \sin \theta_2 & \cos \theta_2 & 0 & a_2 \sin \theta_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (4)$$

$$A \times B = \begin{pmatrix} \cos (\theta_1 + \theta_2) & -\sin (\theta_1 + \theta_2) & 0 & (a_1 \cos \theta_1 + a_2 \cos (\theta_1 + \theta_2)) \\ \sin (\theta_1 + \theta_2) & \cos (\theta_1 + \theta_2) & 0 & (a_1 \sin \theta_1 + a_2 \sin (\theta_1 + \theta_2)) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (5)$$

Sabendo que os dois primeiros elementos da quarta coluna são respectivamente x e y finais, assim já não mais necessitando a equação individual por hastes, com isso diminuindo o tempo de movimentação do braço incluindo apenas as equações finais na programação, observadas nas equações 6 e 7 a seguir:

$$x = a_1 \cos \theta_1 + a_2 \cos (\theta_1 + \theta_2) \quad (6)$$

$$y = a_1 \sin \theta_1 + a_2 \sin (\theta_1 + \theta_2) \quad (7)$$

## 2.7. CONCEITO DE HIL (HARDWARE IN THE LOOP)

*Hardware in The Loop (HIL)* é um conceito criado em 1950 para simulação de uma situação real em um ambiente embarcado. (*National Instruments*, 2016). Foi desenvolvido para substituir ou servir de apoio para análises de um determinado sistema que é muito complexo e robusto.

Utilizando o HIL em laboratório é possível analisar o comportamento de um sistema antes da aplicação final, simular e avaliar defeitos, garantindo a eficácia, qualidade e segurança do projeto que muitas vezes pode ser extremamente caro. (SARHADI, 2013)

O HIL implementado na robótica, os ambientes são possíveis de simulação, operação, podem ser controlados via internet, e os resultados podem ser trabalhados para ser colocada no HIL, a simulação pode ser controlada via software, pelo telefone celular, tudo aquilo que permite uma comunicação, robôs móveis e robôs manipuladores, respectivamente.

Com tecnologia HIL, é possível simular o modelo animado de robôs industriais, validando a estrutura e comportamento do mecanismo, controlando atuadores, realizando simulações e analisando o desenvolvimento. Como justificado pela *National Instruments* (2016), tudo isso se faz possível com o avanço da tecnologia, que proporciona simulações cada mais precisas e próximas a realidade.

A aplicação demonstrada neste projeto sobre a tecnologia HIL é aplicada em um ambiente robótico, entretanto é possível controlar uma diversidade de situações com o conceito, como por exemplo o controle de temperatura, desenvolvimento de produtos e até mesmo simular processos e situações de para uma aeronave.

A tecnologia HIL é um conceito de execução em *real-time*, simulando em tempo real a integração entre *hardware*, *software*, atuadores e sensores. O objetivo é obter uma simulação que seja fiel a realidade e permita observar o comportamento de um sistema sem a necessidade de colocar em risco uma planta ou mecanismo caro e delicado. (*National Instruments*, 2016).

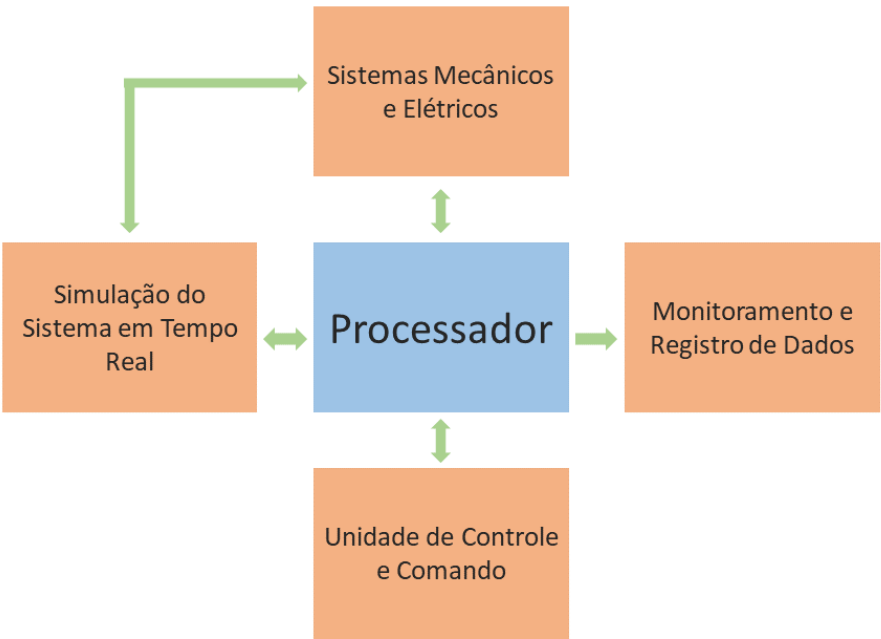
Com a possibilidade de simular elementos do processo, a tecnologia HIL não necessita que todos os componentes do mecanismo sejam físicos e reais, como apresentado por SARHADI (2013), é possível combinar situações de simulação alternando entre elementos físicos e simulados a fim de obter o resultado desejado.

As combinações possíveis e suas nomenclaturas são apresentadas na tabela 3 abaixo, adaptada da e traduzida de SARHADI (2013).

**Tabela 3:** Possibilidades de simulação de um sistema HIL (Traduzido de Sarhadi, 2013).

Linha	Modo	Sensor	Sistema	Atuadores	Computadores
1	Totalmente simulado ou SIL	Simulado	Simulado	Simulado	Simulado
2	Parcialmente simulado ou PIL	Simulado	Simulado	Simulado	Físico
3	Simulado com Atuadores	Simulado	Simulado	Físico	Físico
4	Ideal Hardware in the Loop	Físico	Simulado	Físico	Físico
5	Teste Prático	Físico	Físico	Físico	Físico

Percebe que a tecnologia é maleável e adaptável para o projeto em que se deseja implementar um sistema HIL, contudo, de modo geral temos uma estrutura padrão que auxilia a estruturação do projeto. Uma estrutura simples e análoga a um sistema HIL é observada na Figura 17 a seguir.



**Figura 17:** Estrutura análoga de um sistema HIL (Traduzido de National Instruments, 2016)

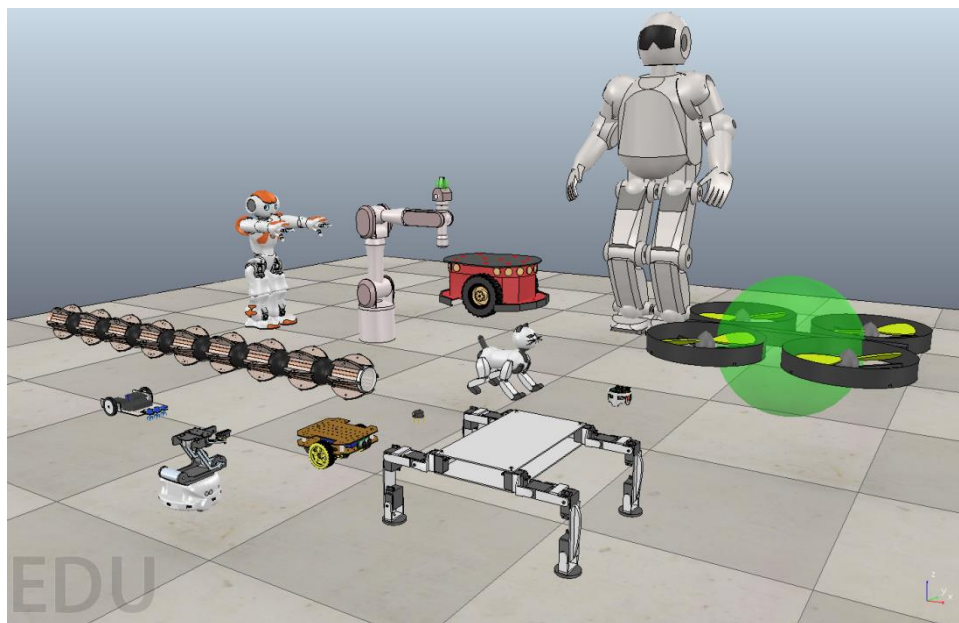
## 2.8. SOFTWARE DE SIMULAÇÃO (V-REP)

V-REP, acrônimo para *Virtual Robot Experimentation Platform*, desenvolvido e publicado pela *Coppelia Robotics*, funciona como um laboratório virtual, para implementação e simulação de sistemas robóticos (*Coppelia Robotics*, 2018).

No software, é possível realizar a simulação de testes de movimentos, teste de força, resistência, etc., permitindo assim a visualização do comportamento de um mecanismo robótico, podendo modificar, programar rotinas, inserir defeitos e simular acontecimentos adversos.

A robótica é um conceito multidisciplinar, abrangendo várias áreas do conhecimento, mecânico, elétrico, eletrônico, da informática e matemática. Pedagogicamente passando para os usuários, desafios e conhecimentos diversos, assim, o software atua integrando áreas de mecânica, programação e matemática.

A Figura 18 abaixo apresenta uma imagem do ambiente de simulação do software com alguns robôs carregados e adicionados a uma cena.



**Figura 18:** Visão do software V-Rep, com alguns robôs disponíveis (*Coppelia Robotics*, 2018).

O software V-Rep apresenta desde robôs fixos como móveis, permitindo a visualização de diversos tipos de movimentação, desde robôs com 2 graus de liberdade, até robôs homini-direcionais, trabalhando direções, obstáculos e caminhos a serem percorridos (*Coppelia Robotics*, 2018).

### 2.8.1. MOTORES DE SIMULAÇÃO

Para a execução de cálculos e simulação de elementos de maneira realística, o software conta com quatro tipos de motores de simulação de física, isto é, ambientes programados que simulam as condições reais de interação entre corpos e materiais, considerando fatores como gravidade, atrito, peso, velocidade entre outros.

Os motores disponíveis são gratuitos e de código aberto, *Bullet Engine*, *ODE (Open Dynamics Engine)*, *Vortex Engine* e *Newton Dynamics*. Como define a *Bullet Engine* (2018), os motores de físicas aplicados ao software permitem que as simulações sejam pautadas em condições físicas reais.

Assim, o robô simulado pode colidir com outros corpos, sofrer deformação ou até mesmo ser destruído, fazendo com que a programação de movimento seja o mais próximo da realidade possível (*Coppelia Robotics*, 2018).

### 2.8.2. PROGRAMAÇÃO EM LUA

O software V-Rep é bastante flexível quanto a customização das simulações criadas, apresentando mais de uma opção quanto a programação dos modelos robóticos simulados (*Coppelia Robotics*, 2018).

A maneira padrão de programação de modelos no V-Rep, é através de scripts na linguagem de programação Lua, que como definido pela Lua (2018), é uma linguagem leve e feita com a finalidade de integrar softwares escritos em diferentes linguagens.

A programação em Lua é feita dentro do software V-Rep, ou integrada a simulação no momento da execução. Caso o software escrito precise ser executado externamente ao V-Rep, como por exemplo, em um robô ou hardware real, é possível utilizar as funcionalidades disponibilizadas pela API do V-Rep.

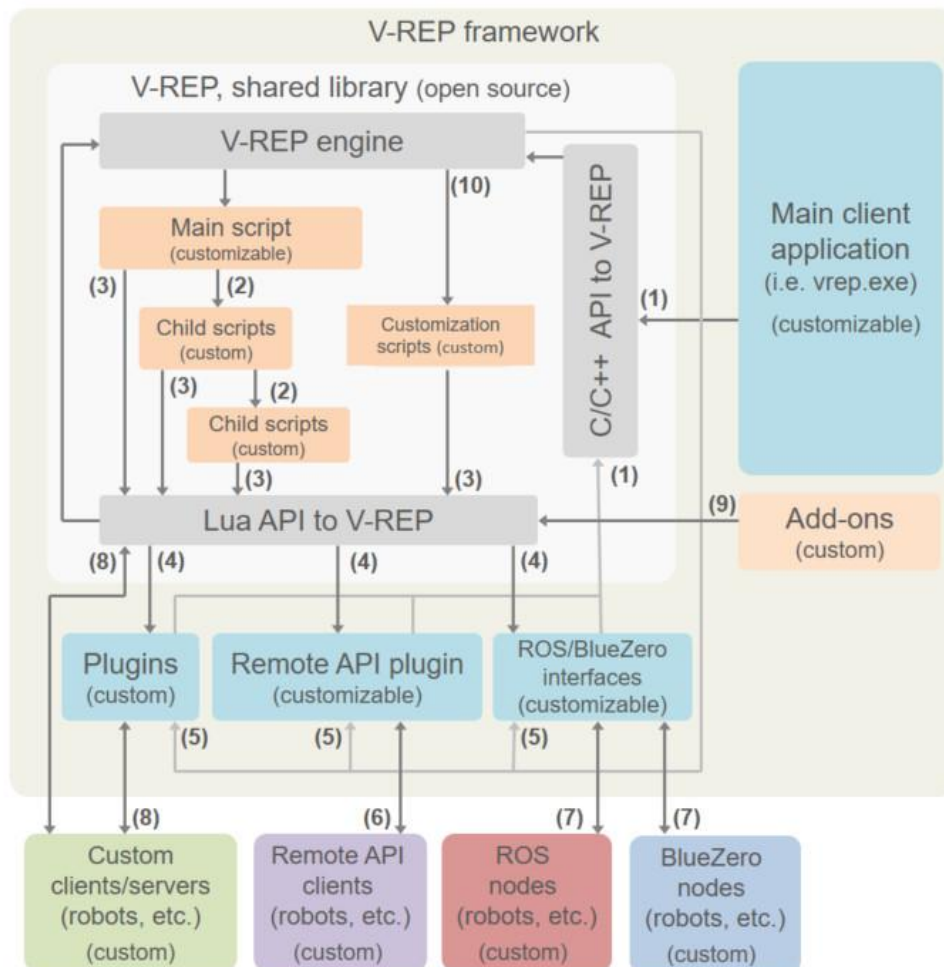


### 2.8.3. V-REP API

Para comunicar o *software* com elementos externos ao V-Rep, é necessário utilizar a API (*Application Programming Interface*) provida pela *Coppelia Robotics*, que é definida pela documentação da Coppelia Robotics (2018) por um conjunto de funções que interagem com a simulação e podem ser integradas em diferentes plataformas e linguagens de programação.

Podem ser implementados clientes para a aplicação com diversas linguagens suportadas, *C/C++*, *Python*, *Matlab*, *Octave* e *Lua*. Para o desenvolvimento do programa foi desenvolvido um cliente na linguagem *Python*.

Na Figura 19 é apresentada a arquitetura do framework do V-Rep, onde é possível observar os diferentes níveis de abstração do software, em níveis de código executado no V-Rep (scripts em Lua) e código executado em hardware alheio a simulação (API).



**Figura 19:** Diagrama estrutural da V-Rep API (Coppelia Robotics, 2018).

## 2.9. PLATAFORMA ARDUINO

O *Arduino* é uma plataforma de prototipagem eletrônica, de hardware e software livre, que apresenta um microcontrolador *Atmel* embarcado permitindo ligamento de motores, *LEDS*, *sensores*, *displays* e *shields*. (Arduino, 2018).

A comunidade *Arduino*, vem crescendo de maneira exponencial, desde sua criação, pois de maneira fácil e eficiente criam programas, componentes e situações em que pode ser implementado, como controle de processos industriais simples, automação residencial, trabalhos com robótica, tanto móvel como industrial, controle de válvulas, etc.

Existem diversas placas da família *Arduino*, desde as *Nano*, até as configurações *Mega2560*, todos contando com um *chip-set*, que conforme explicado pela *Arduino* (2018), podem variar sua capacidade de armazenamento interno, e processamento. A Figura 20 mostra a representação de uma placa *Arduino* do tipo *Uno*.

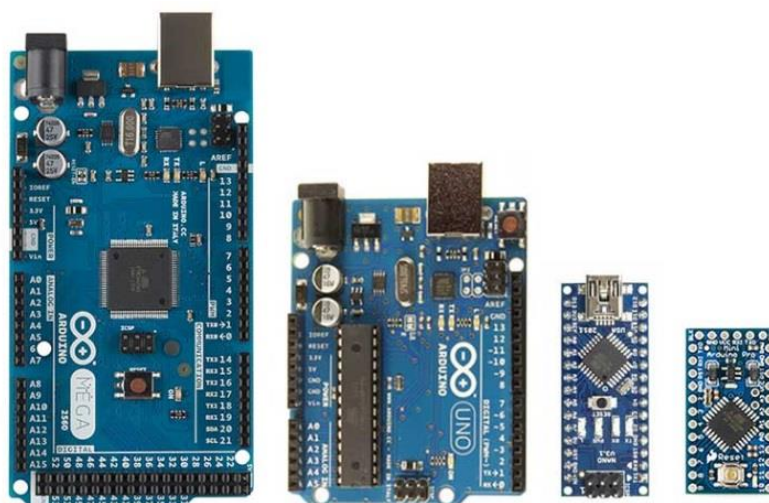


**Figura 20:** Placa Arduino Uno (Arduino, 2018).

Por ser um projeto de *software* livre, o *Arduino* pode ser utilizado em plataformas *Mac*, *Windows* e *Linux*, todos compartilhando novas bibliotecas por suas diversas comunidades de desenvolvedores. Sendo barato e preciso, além de rápido e eficaz, o *Arduino* está sendo a escolha de diversos projetos no âmbito da tecnologia por todas essas facilidades.

De hardware completo e simples temos a versão mais comum do *Arduino* sendo a placa *UNO*, outras são bem parecidas havendo apenas algumas variações de memória ROM/RAM, processamento e preço.

A Figura 21 abaixo apresenta alguns tipos de placa *Arduino* lado a lado, da esquerda para a direita *Arduino Mega*, *Arduino Uno*, *Arduino Nano*, *Arduino Mini*.



**Figura 21:** Família de placas *Arduino* (Arduino, 2018).

## 2.10. PROGRAMAÇÃO DE ROBÔS

Com o crescimento do desenvolvimento dos processos de manufatura, foi necessário introduzir a ideia de que a produção fosse flexibilizada, tanto na quantidade quanto na variedade de produtos a serem produzidos. (GROOVER, 1998).

Nesse processo de desenvolvimento surge o conceito de automação flexível, onde basicamente o layout existente no chão de fábrica, é feito de forma a flexibilizar a produção, se adaptando rapidamente a diferentes produtos ou processos. (NIKU, 2013).

A necessidade de rapidamente reconfigurar uma linha de produção para iniciar o desenvolvimento de outro produto ou processo, passou a exigir que as ferramentas utilizadas fossem de simples reconfiguração, surge então o conceito de *reprogramabilidade*, introduzido por Charles Devol, que desenvolveu métodos de registrar sequencias de movimentos programáveis em seus dispositivos, iniciando assim a primeira geração de robôs, como explicado por GROOVER (1998).

Usualmente, existe uma interface entre o operador e o robô, com linguagens de programação legíveis a humanos que se traduzem em movimentos a serem interpretados pelo mecanismo robótico. A Figura 22 a seguir mostra um dispositivo de interface de comunicação com um robô articulado da fabricante KUKA.



**Figura 22:** Interface de um Robô industrial KUKA (KUKA, 2018).

## 2.11. MÉTODOS DE PROGRAMAÇÃO

Para NIKU (2013), robôs industriais de hoje, de modo geral, são mecanismos automatizados que são projetados a fim de movimentar peças ou ferramentas sobre uma trajetória previamente definida, de modo a realizar uma tarefa desejada.

O programa do robô deverá ser capaz de se adaptar a diferentes situações desejadas, alterando funções de operação, movimentação e percurso de acordo com o processo a ser realizado (GROOVER, 1998). A flexibilidade de um robô é avaliada pela extensão do tipo de operações que podem ser realizadas, bem como os movimentos que podem ser programados em seu controlador.

A programação de um mecanismo robótico pode ser feita de duas maneiras: *Programação on-line*, e *Programação off-line*:

- **Programação on-line** é aquela que é realizada de fato com o robô físico e implantando em um ambiente real, e pode ser feita utilizando técnicas de condução, aprendizagem ou linguagem textual.
- **Programação off-line** por sua vez, é realizada sem o uso do robô real, e sim com um modelo virtual simulando o ambiente final da aplicação.

Segundo ROSÁRIO (2004), a eficiência de um método de programação de um mecanismo robótico é medida levando em consideração o tempo gasto para implantar o sistema em funcionamento.

De modo geral, duas abordagens são utilizadas para programar a movimentação de mecanismos robóticos, quanto ao caminho a ser percorrido pelo robô: movimento ponto a ponto e movimento contínuo.

- **Movimento ponto a ponto:** o robô se move de um ponto a outro, sem considerar a trajetória desenvolvida, podendo realizar otimizações, controlando da maneira que julgar mais adequada a velocidade de cada grau de liberdade do mecanismo.
- **Movimento contínuo:** movimenta-se de um ponto a outro, mas considerando a trajetória desenvolvida, armazenando os incrementos necessários em cada eixo para percorrer o percurso de um ponto a outro.

## **2.12. PROGRAMAÇÃO ON-LINE**

Na programação do tipo on-line, a rotina a ser executada pelo mecanismo robótico é construída utilizando o robô real em um ambiente de aplicação ou controlado, como definido por ROSÁRIO (2004). Pode ser realizada por técnicas de aprendizagem ou utilizando linguagem textual de programação.

### **2.12.1. PROGRAMAÇÃO POR APRENDIZAGEM**

Na programação por aprendizagem, o programador conduz o robô para a trajetória que deseja ser executada, de modo que o robô “aprenda” qual caminho percorrer.

Uma vez gravados os pontos referentes à trajetória a ser percorrida e instruções a ser realizado, o robô é capaz de reproduzir os comandos inseridos e entrar em operação.

### **2.12.2. PROGRAMAÇÃO POR LINGUAGEM TEXTUAL**

A programação por linguagem textual se diferente da por aprendizagem pelo fato de que o robô não é conduzido até os pontos críticos de sua trajetória, todo o processo é realizado através de linguagem de programação, com comandos específicos e suportados pelo robô.

Como apresentado por ROSÁRIO (2004), durante o processo de programação on-line, tem-se a desvantagem de que o robô fica inutilizado para operação durante a programação. O tempo em que o robô se encontra parado é um fator crítico para determinar a produtividade e flexibilidade do método de programação e operação do robô.

## **2.13. PROGRAMAÇÃO OFF-LINE**

O desenvolvimento da tecnologia empregada na produção de robôs, tanto em *hardware* quanto em *software*, tem feito com que seja cada vez mais viável a utilização da programação *off-line* para mecanismos robóticos.

Como definido por NIKU (2013), conceitualmente, a programação *off-line* consiste em utilizar modelos virtuais e simulados dos robôs atuando em situações reais, utilizando a programação textual que seria de fato aplicada na instalação, e observando sua atuação em um ambiente simulado.

### 3. MATERIAIS E MÉTODOS

#### 3.1. ESTRUTURA DO PROJETO

O desenvolvimento do modelo matemático e simulação do robô do tipo SCARA, se deram com a integração de tecnologias diferentes em ambientes diferentes. O objetivo geral foi desenvolver um modelo matemático, capaz de descrever os movimentos do manipulador, e integra-lo em tecnologia *hardware in the loop*, utilizando software de simulação para observar visualmente as respostas do sistema implantado.

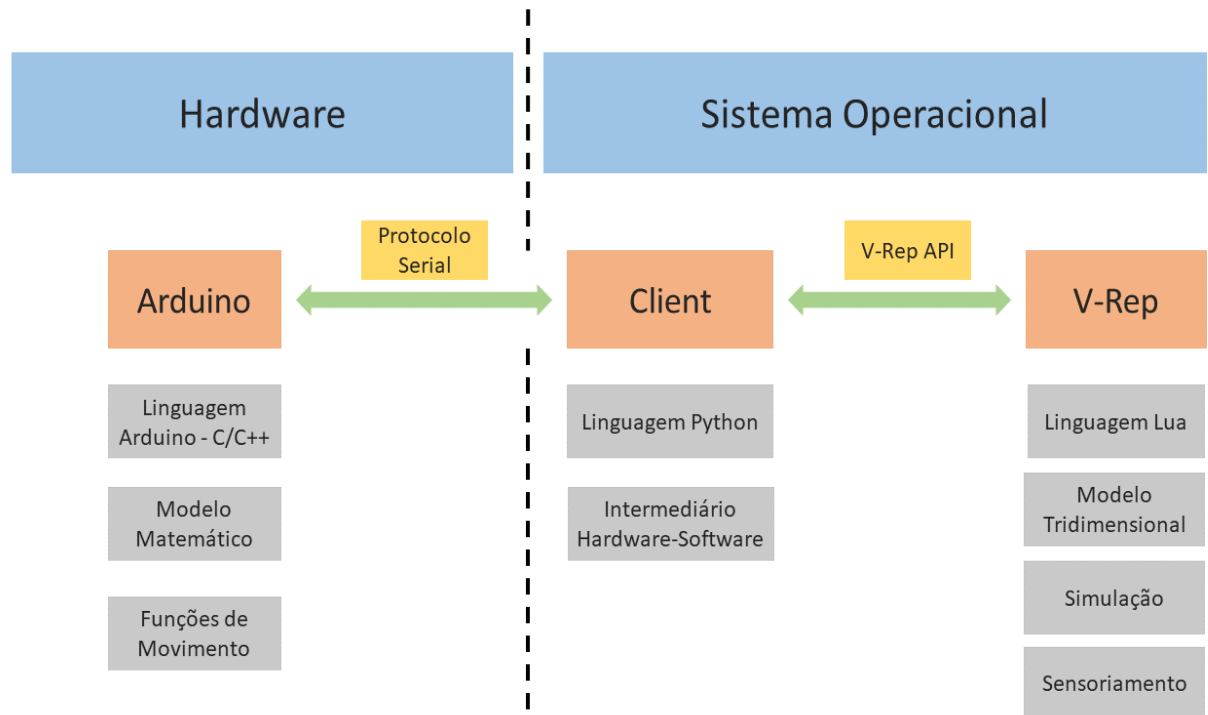
De modo geral, as etapas de desenvolvimento se dividem da seguinte forma:

- I. Desenvolver modelo matemático e teórico, definindo as equações que descrevem os movimentos do manipulador;
- II. Aplicar modelo matemático em uma linguagem de programação a fim de verificar a veracidade das equações desenvolvidas;
- III. Definir modelo tridimensional do manipulador no ambiente de simulação a ser utilizado (V-Rep);
- IV. Aplicar modelo matemático em linguagem de programação embarcada no ambiente *hardware in the loop* a ser utilizado (Arduino);
- V. Desenvolver comunicação entre ambiente *hardware in the loop* e ambiente de simulação;
- VI. Simular diferentes rotinas de movimentação com o manipulador robótico.

Para a etapa de aplicação do modelo matemático em linguagem de programação, a fim de verificar a veracidade das equações desenvolvidas, foi utilizado a linguagem Python, e para a linguagem de programação no ambiente embarcado, foi utilizado a própria linguagem da plataforma escolhida, o Arduino, que é uma linguagem baseada em C/C++.

A comunicação entre o ambiente *hardware in the loop* e o ambiente de simulação, se deu através do protocolo de comunicação *serial* e todas as simulações e etapas foram realizadas em um computador utilizando sistema operacional *Windows*. Para o ambiente de simulação, foram aplicadas rotinas programadas na linguagem *Lua*, que é a linguagem padrão do *software* V-Rep.

A comunicação com o ambiente simulado no *software* V-Rep, precisa ser feita através das funções da API disponibilizada. Para isso, foi escrito um *client* em Python, ou seja, um programa que atua como intermediário na comunicação entre o *hardware* e a simulação. O esquema geral da estrutura do projeto pode ser observado na Figura 23 abaixo.



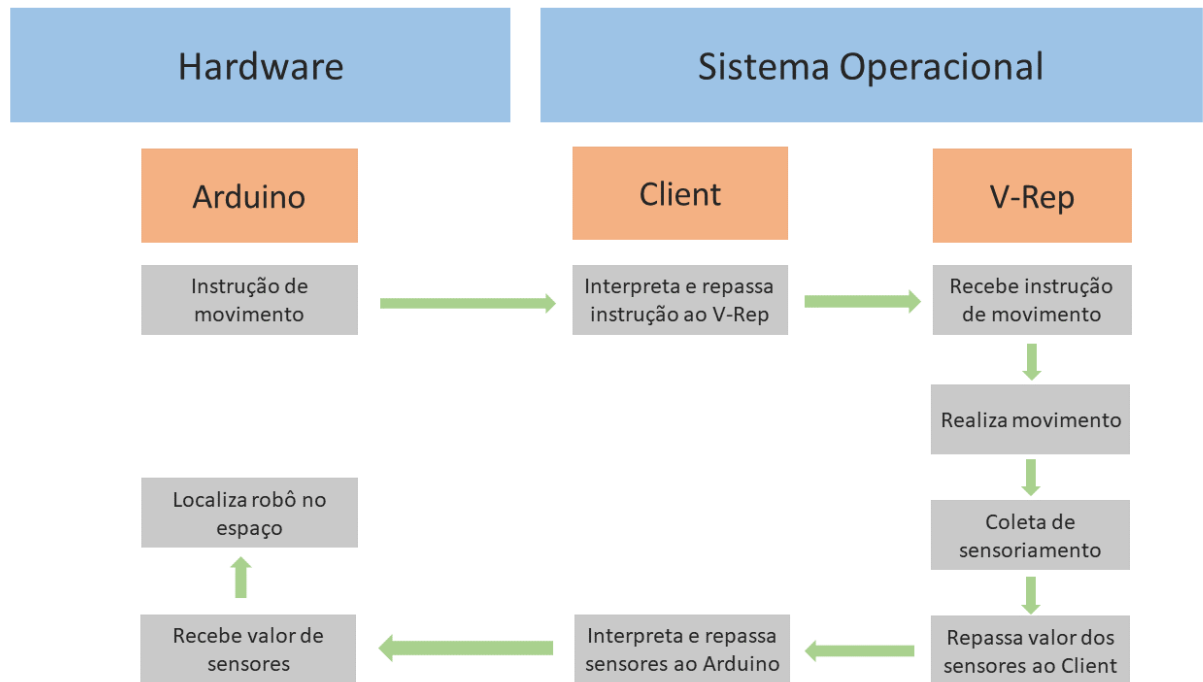
**Figura 23:** Arquitetura geral do projeto.

O projeto divide-se em basicamente duas partes, *hardware* e *software*, sendo a primeira parte a plataforma embarcada (Arduino) e a segunda o sistema operacional utilizado (*Windows*), contendo o *client* e o *software* de simulação V-Rep.

No Arduino ficam os códigos responsáveis por descrever o modelo matemático e as funções de movimento do robô. No *client* ficam as rotinas responsáveis por se comunicar com o Arduino via protocolo Serial e repassar informações para o simulador através da V-Rep API.



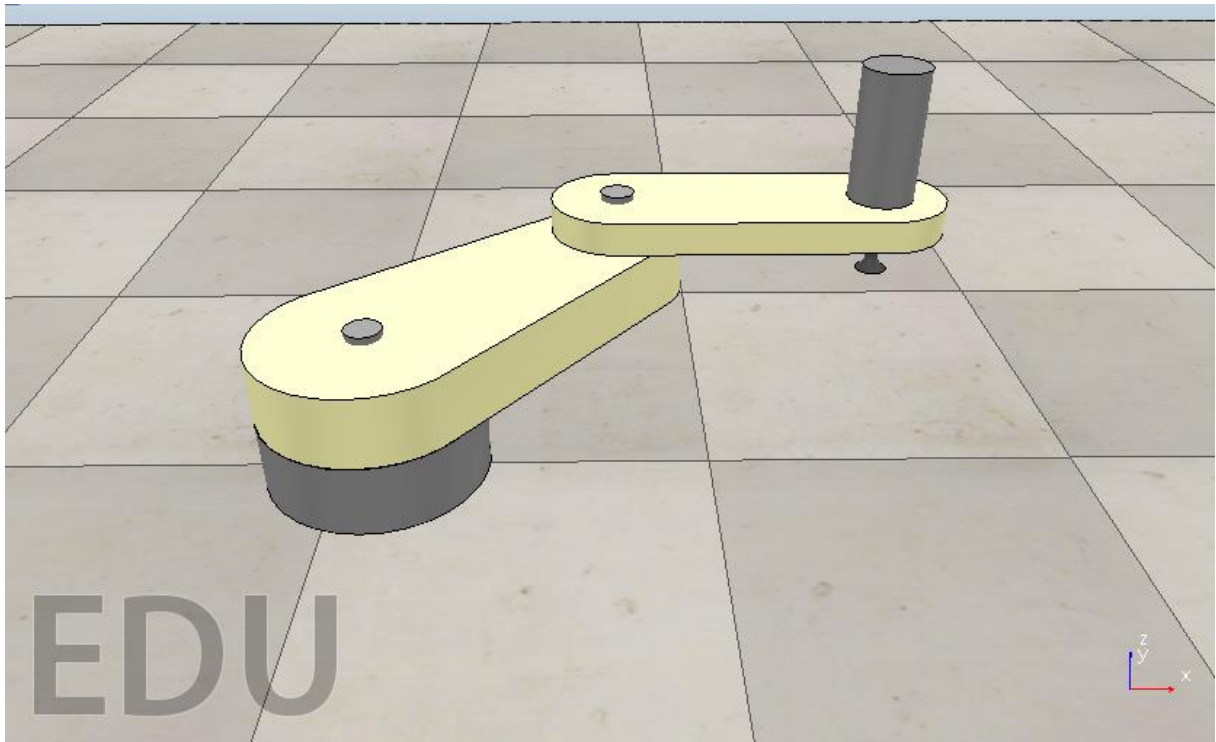
O *client* tem também a função de repassar as informações de sensoriamento do modelo simulado no V-Rep para o Arduino. Em um ambiente real, os sensores instalados no robô seriam responsáveis por dizer ao controlador onde o mesmo se encontra no espaço, já em um ambiente simulado, são utilizadas as leituras de sensoriamento do próprio *software* de simulação.



**Figura 24:** Estrutura de um ciclo de comunicação de movimento.

A Figura 24 acima apresenta de maneira geral a rotina realizada durante a simulação do robô. Os valores de sensores colhidos pelo V-Rep são enviados ao Arduino para representar a leitura de um sensor real, o robô então se localiza no espaço e dá procedimento a rotina de movimentação.

Para a simulação, foi utilizado um modelo tridimensional de um manipulador SCARA disponibilizado no *software* V-Rep. O modelo conta com as duas juntas rotacionais dos elos 1 e 2, e com uma junta prismática na extremidade, com uma ferramenta do tipo sucção, como uma ventosa. O robô utilizado é apresentado na Figura 25 a seguir.



**Figura 25:** Modelo Tridimensional do robô SCARA, no software V-Rep.

Durante a simulação, o robô pode interagir com outros elementos tridimensionais, simulando colisões e a própria ação de pegar ou soltar um objeto. Assim, é possível programar rotinas de montagens ou movimentação de peças para demonstrar a precisão e funcionalidade do modelo desenvolvido.

Para seguir com o desenvolvimento do projeto, a primeira etapa é realizar a modelagem matemática do sistema, seguindo da verificação do modelo com uma linguagem de programação.

### 3.2. MODELAGEM MATEMÁTICA

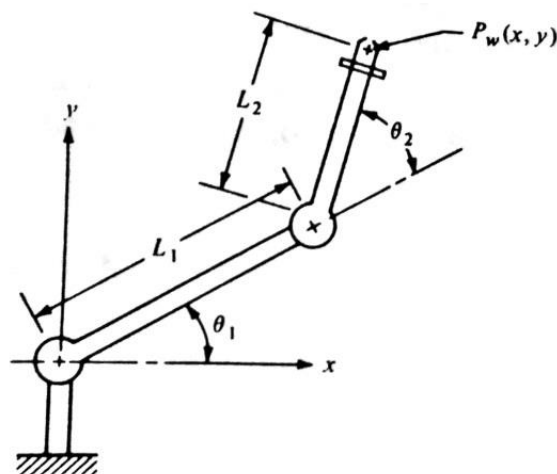
Para desenvolver um método de controlar o movimento de um manipulador robótico, é necessário descrever o movimento do robô matematicamente, isto é, definir quais equações descreve o movimento a ser realizado, localizando o robô no espaço.

As variáveis que irão descrever o movimento de um mecanismo robótico, em específico o manipulador SCARA, de maneira geral, englobam comprimento dos elos, ângulos de rotação e coordenadas desejadas. Para definir o modelo matemático do manipulador, foram feitas duas abordagens, um modelo matricial e um modelo trigonométrico.

#### 3.2.1. MODELAGEM MATRICIAL

Como citado anteriormente, o movimento do robô poder feito utilizando das técnicas de cinemática, que incluem cinemática inversa e cinemática direta, realizando a passagem do espaço cartesiano para o espaço das juntas e do espaço das juntas para o espaço cartesiano, respectivamente.

Para desenvolver os modelos matemáticos, foi tomada como esquema a Figura 26, que ilustra as nomenclaturas de ângulos e comprimento de elos do manipulador SCARA.



**Figura 26:** Modelo básico de um manipulador SCARA (GROOVER, 1998).

Desse modo, o ângulo da junta rotacional da base é referido como  $\theta_1$ , enquanto o da junta rotacional entre os elos é referido como  $\theta_2$ . Já os comprimentos dos elos são tratados como  $l_1$  e  $l_2$ . Com essas informações, é possível definir os elementos de Denavit-Hartenberg, como observado na Tabela 4 a seguir.

**Tabela 4:** Tabela de aquisição de dados Denavit-Hartenberg.

<i>Ligamento</i>	$a_i$	$\alpha_i$	$d_i$	$\theta_i$
1	$l_1$	0	0	$\theta_1$
2	$l_2$	0	0	$\theta_2$

É possível observar que cada elo do manipulador SCARA aponta para uma posição no espaço, sendo assim, considera-se cada elo do robô como sendo um vetor, que aponta para uma posição com uma direção  $\theta$ , observa-se também que de tal modo, o vetor correspondente ao elo 2 tem como referência o vetor do elo 1.

Representando cada vetor, ou seja, cada elo, como uma matriz de referência, é possível localizar o elo no espaço, indicando sua origem e sua posição final. Para fins de cálculo de projeto, a origem do sistema foi considerada no início do elo 1.

Os elementos contidos nas matrizes são definidos utilizando propriedades trigonométricas para localização dos vetores. Sendo assim, as equações 8 e 9 indicam as matrizes de referência que representam os elos 1 e 2, representados como matrizes A e B, respectivamente.

$$A = \begin{pmatrix} \cos(\theta_1) & -\sin(\theta_1) & 0 & l_1 \times \cos(\theta_1) \\ \sin(\theta_1) & \cos(\theta_1) & 0 & l_1 \times \sin(\theta_1) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (8)$$

$$B = \begin{pmatrix} \cos(\theta_2) & -\sin(\theta_2) & 0 & l_2 \times \cos(\theta_2) \\ \sin(\theta_2) & \cos(\theta_2) & 0 & l_2 \times \sin(\theta_2) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (9)$$

Utilizando a definição de que o vetor correspondente ao elo 2 inicia-se no fim do elo 1, ou seja, tem como referência o elo 1, a operação vetorial que representa um terceiro vetor correspondente a posição final do sistema, é definido como a multiplicação das matrizes que representam os dois vetores.

A multiplicação vetorial refletida nas matrizes gera uma terceira matriz, contendo a posição final do atuador, que nada mais é do que a indicação de um terceiro vetor correspondente à posição da extremidade do atuador. A matriz contendo a multiplicação é observada na equação 10 a seguir.

$$A \times B = \begin{pmatrix} \cos(\theta_1 + \theta_2) & -\sin(\theta_1 + \theta_2) & 0 & l_1 \times \cos(\theta_1) + l_2 \times \cos(\theta_1 + \theta_2) \\ \sin(\theta_1 + \theta_2) & \cos(\theta_1 + \theta_2) & 0 & l_1 \times \sin(\theta_1) + l_2 \times \sin(\theta_1 + \theta_2) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (10)$$

Sabendo que a matriz  $A \times B$  corresponde a um vetor resultante construído a partir da multiplicação dos vetores dos elos e representa a posição final do atuador, as coordenadas finais do atuador estão contidas nesta mesma matriz, mais precisamente na quarta coluna da matriz. Deste modo, é possível resumir a transformação do braço com a cinemática inversa utilizando estes termos, observados nas equações 11 e 12.

$$x = l_1 \times \cos(\theta_1) + l_2 \times \cos(\theta_1 + \theta_2) \quad (11)$$

$$y = l_1 \times \sin(\theta_1) + l_2 \times \sin(\theta_1 + \theta_2) \quad (12)$$

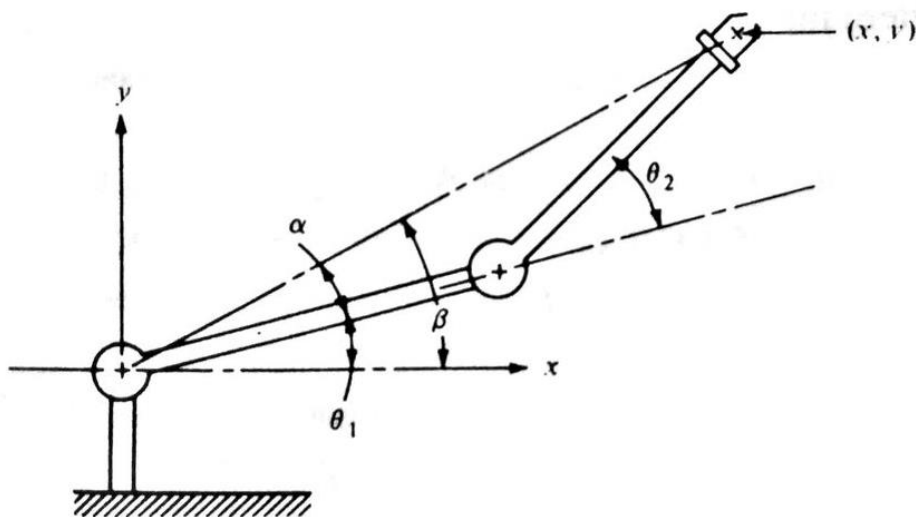
Computacionalmente, executar funções para calcular a formula final para um modelo generalizado é melhor do que recalcular a multiplicação matricial a cada instante que se deseja realizar um movimento, portanto, as fórmulas acima são as que serão implementadas nos modelos programados.

### 3.2.2. MODELAGEM TRIGONOMÉTRICA

Além da modelagem matricial, é possível desenvolver pra alguns modelos de robôs, um modelo trigonométrico. O robô do tipo SCARA especialmente apresenta um modelo trigonométrico simples de ser desenvolvido.

Tomando novamente o esquema da Figura 26 como representação do manipulador utilizado, foram utilizadas as mesmas nomenclaturas para descrever modelagem do manipulador.

De modo geral, o modelo trigonométrico objetiva demonstrar o posicionamento do manipulador SCARA espacialmente com propriedades trigonométricas, traçando um triângulo entre a origem do robô, os elos e a extremidade, conforme observado na Figura 27 a seguir.



**Figura 27:** Robô SCARA em modelo trigonométrico (GROOVER, 1998).

Assim, a modelagem trigonométrica para realizar a cinemática inversa, visa relacionar matematicamente através de propriedades trigonométricas a posição final do atuador  $(x, y)$  em função dos ângulos aplicados nas juntas ( $\theta_1$  e  $\theta_2$ ), de modo a resolver todas as equações sem a utilização de matrizes de referencia como no modelo anterior.

Para desenvolver a cinemática direta do robô SCARA, isto é, descobrir as coordenadas da extremidade do manipulador em função dos ângulos conhecidos nas juntas, novamente é utilizado o conceito de elos representados por vetores, sendo assim, podemos definir que a posição final do braço no espaço cartesiano é definida por dois “vetores”, um para o elo 1 e outro para o elo 2.

$$v = [x, y] \quad (13)$$

$$v1 = [l1 \cos(\theta1), \quad l1 \times \text{sen}(\theta1)] \quad (14)$$

$$v2 = [l2 \cos(\theta1 + \theta2), \quad l2 \times \text{sen}(\theta1 + \theta2)] \quad (15)$$

Novamente, a soma vetorial dos vetores gera as coordenadas  $x$  e  $y$  da extremidade do braço para o (ponto  $P_w$ ) no espaço cartesiano. Nota-se que a equação 16 resultante, é a mesma resultante do processo matricial.

$$v3 = [v1] + [v2] \quad (16)$$

$$v3 = [l1 \times \cos(\theta1) + l2 \times \cos(\theta1 + \theta2), l1 \times \text{sen}(\theta1) + l2 \times \text{sen}(\theta1 + \theta2)] \quad (16.1)$$

Extraindo as coordenadas  $x$  e  $y$  do vetor resultante  $v3$ , se obtém a expressão que relaciona a coordenada do manipulador em função dos valores conhecidos dos ângulos aplicado nas juntas.

$$x = l1 \times \cos(\theta1) + l2 \times \cos(\theta1 + \theta2) \quad (17)$$

$$y = l1 \times \text{sen}(\theta1) + l2 \times \text{sen}(\theta1 + \theta2) \quad (18)$$

As equações 17 e 18 apresentam a posição do manipulador no espaço, partindo do pressuposto de que os ângulos aplicados nas juntas são conhecidos, ou seja, realiza a cinemática direta. Para realizar a cinemática inversa, é preciso desenvolver matematicamente a expressão para colocar as equações em função das coordenadas  $x$  e  $y$ .

A fim de desenvolver as expressões, foram utilizadas propriedades trigonométricas para desenvolver o cálculo das funções cosseno e seno de uma soma de ângulos.

$$\cos(A + B) = \cos(A) \times \cos(B) - \sin(A) \times \sin(B) \quad (19)$$

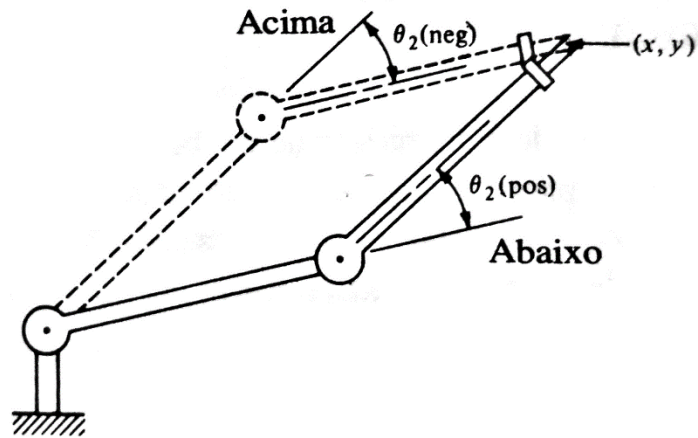
$$\sin(A + B) = \sin(A) \times \cos(B) + \sin(B) \times \cos(A) \quad (20)$$

Utilizando as propriedades, é possível reescrever as equações 21 e 22 como:

$$x = l_1 \times \cos(\theta_1) + l_2 \times \cos(\theta_1) \times \cos(\theta_2) - l_2 \times \sin(\theta_1) \times \sin(\theta_2) \quad (21)$$

$$y = l_1 \times \sin(\theta_1) + l_2 \times \sin(\theta_1) \times \cos(\theta_2) + l_2 \times \cos(\theta_1) \times \sin(\theta_2) \quad (22)$$

No manipulador do tipo SCARA, cada posição no espaço pode ser acessada utilizando duas configurações possíveis de juntas, como observado na Figura 28, isso faz com que existam dois valores possíveis para o ângulo  $\theta_2$ .



**Figura 28:** Dualidade de acesso a uma posição no espaço (GROOVER, 1998).



Para fins de generalização da fórmula, as equações a seguir consideram o ângulo na segunda junta como positivo. Continuando o desenvolvimento das equações, elevando ambas ao quadrado e somando as duas, obtém-se:

$$\cos(\theta_2) = \frac{x^2 + y^2 - l_1^2 - l_2^2}{2 \times l_1 \times l_2} \quad (23)$$

Conforme observado na Figura 28, é necessário definir os valores dos ângulos  $\alpha$  e  $\beta$  a fim de continuar com o cálculos dos ângulos  $\theta_1$  e  $\theta_2$ .

$$\operatorname{tg} \alpha = \frac{l_2 \times \operatorname{sen}(\theta_2)}{l_2 \times \cos(\theta_2 + l_1)} \quad (24)$$

$$\operatorname{tg} \beta = \frac{y}{x} \quad (25)$$

Nesta etapa, utiliza-se mais uma identidade trigonométrica para resolver a tangente do ângulo  $\alpha$  para obter a equação 26.

$$\operatorname{tg}(A - B) = \frac{\operatorname{tg}(A) - \operatorname{tg}(B)}{1 + \operatorname{tg}(A) \times \operatorname{tg}(B)} \quad (26)$$

$$\operatorname{tg}(\theta_1) = \frac{[y \times (l_1 + l_2 \times \cos(\theta_2)) - x \times l_2 \times \operatorname{sen}(\theta_2)]}{[x(l_1 + l_2 \cos(\theta_2)) + y \times l_2 \times \operatorname{sen}(\theta_2)]} \quad (27)$$

Agora, conhecendo os comprimentos dos elos  $l_1$  e  $l_2$ , pode-se calcular os ângulos requeridos das juntas para colocar o braço numa posição  $(x, y)$  no espaço cartesiano. As equações 23 e 27 são as que serão implementadas utilizando linguagem de programação para realizar o movimento do braço robótico.

### 3.3. MODELADEM EM PYTHON

Após definir as fórmulas a serem utilizadas conforme o modelo matemático demonstrado, a etapa seguinte foi o desenvolvimento de um primeiro modelo programado na linguagem Python.

Este modelo foi utilizado para colocar o conceito em prova e verificar a confiabilidade das equações obtidas, utilizando de gráficos e recursos visuais para demonstrar a utilização das equações de cinemática inversa e direta.

Vale ressaltar, que o modelo em Python neste momento ainda não apresenta ligação com a simulação do robô, e sim apenas com a visualização prévia das funções de cinemática e visualização do *workspace* do robô a ser desenvolvido.

#### 3.3.1 BIBLIOTECAS E DEPENDÊNCIAS

Durante o desenvolvimento do modelo cinemático em Python, foram utilizadas algumas bibliotecas para auxiliar com funções principalmente matemáticas e de visualização gráfica.

Para realizar a plotagem dos gráficos, foi utilizada a biblioteca *Matplotlib*, e para funções matemáticas foi utilizada a biblioteca científica *Numpy*. O trecho de código abaixo em Python mostra o cabeçalho com as bibliotecas utilizadas.

```
1. # -*- coding: utf-8 -*-  
2. import matplotlib.pyplot as plt  
3. import numpy as np  
4. from numpy import cos, sin, arccos, arctan2, power  
5. from numpy import radians, degrees, array
```

Todas as bibliotecas utilizadas, bem como a linguagem Python, são de acesso gratuito e podem ser adquiridas através dos sites oficiais, e contem ampla documentação e comunidade ativa, disponíveis para todas as plataformas *Windows*, *Linux* e *Mac*.

### 3.2.1. CINEMÁTICA DIRETA EM PYTHON

Após as fórmulas serem definidas, foi possível programar funções para realizar as transformações cinemáticas conforme os parâmetros fornecidos. Foram programadas funções para realizar a cinemática inversa e direta do robô.

Abaixo observa-se o código para a função de cinemática direta. Nesta função, é realizada a passagem do espaço das juntas para o espaço cartesiano, utilizando as equações demonstradas anteriormente.

```
1. #Transformação Direta
2. #Passagem do Espaço das Juntas p/ o Espaço Cartesiano
3. def direct_transform(t1, t2, l1, l2):
4.
5.     #Computa vetores individuais para elos L1 e L2
6.     r1 = array([l1 * cos(radians(t1)), l1 * sin(radians(t1))])
7.     r2 = array([l2 * cos(radians(t1 + t2)), l2 * sin(radians(t1 + t2))])
8.
9.     #Soma os vetores individuais p/ obter coordenadas do ponto P(x)
10.    r3 = array(r1 + r2)
11.
12.    #Retorna vetores individuais r1, r2 e vetor final r3
13.    return [r1, r2, r3]
```

Nesta função, são recebidos como parâmetros os ângulos nas juntas rotacionais  $t1$  e  $t2$  em graus, bem como os comprimentos dos elos  $l1$  e  $l2$  em metros, a resposta em forma de um vetor de três posições que contém a posição calculada dos elos individuais  $r1$  e  $r2$ , e a posição final do atuador  $r3$ , ambas em coordenadas cartesianas  $x$  e  $y$ .

É possível notar que no código há conversões realizadas entre graus e radianos. As fórmulas demonstradas utilizam todas as unidades de ângulo como graus, mas computacionalmente a maioria das funções matemáticas utilizam as unidades de ângulo como radianos, assim é necessário adaptar os valores utilizados durante a função.

### 3.3.2. CINEMÁTICA INVERSA EM PYTHON

Na cinemática inversa, é realizada a passagem contrária à cinemática direta, ou seja, é feita a passagem do espaço cartesiano para o espaço das juntas. As equações que fundamentam esta passagem encontram-se demonstradas nas equações 23 e 27.

O código a seguir, apresenta a função de cinemática inversa, no qual são recebidos como parâmetros às coordenadas do ponto cartesiano  $x$  e  $y$ , e os comprimentos dos elos  $l1$  e  $l2$ .

```
1. #Transformação Inversa
2. #Passagem do Espaço Cartesiano para o Espaço das Juntas
3. def inverse_transform(x, y, l1, l2):
4.
5.     #Vetor para armazenar os resultados finais
6.     res = []
7.
8.     #Inicia cálculos para multiplicador -1 e 1
9.     for i in (-1, 1):
10.
11.         #Computa valor de t2
12.         t2=arccos((power(x,2)+power(y,2)-power(l1,2)-power(l2,2))/(2*l1*l2))
13.
14.         #Atualiza multiplicador de t2
15.         t2 *= i
16.
17.         #Verifica se t2 retornou NaN
18.         if np.isnan(t2): t2 = 0
19.
20.         #Computa valor de t1
21.         num = y * (l1 + l2 * cos(t2)) - x * l2 * sin(t2)
22.         den = x * (l1 + l2 * cos(t2)) + y * l2 * sin(t2)
23.         t1 = arctan2(num, den)
24.
25.         #Converte em graus
26.         t1 = degrees(t1)
27.         t2 = degrees(t2)
28.
29.         #Adiciona resposta ao vetor de saída
30.         res.append(t1)
31.         res.append(t2)
32.
33.     #Retorna os ângulos possíveis
34.     return res
```

É importante notar que para a configuração de um manipulador do tipo SCARA, existem duas maneiras de alcançar um determinado ponto dentro da área de trabalho do robô. A Figura 28 apresentada anteriormente ilustra a situação, demonstrando configurações diferentes das juntas que levam a extremidade do braço para a mesma posição.

Assim, foi utilizado no código um multiplicador, para realizar os dois cálculos quando solicitado a função de transformação inversa, o multiplicador alterna seu valor de -1 para 1, calcula as duas possíveis configurações de acesso ao ponto no espaço cartesiano.

Deste modo, a função retorna um vetor de 4 posições, onde cada par de valores representa uma configuração de juntas  $t1$  e  $t2$  para acesso ao ponto no espaço dentro da área de trabalho.

### **3.3.3. TESTANDO O MODELO PROGRAMADO**

Com as funções de cinemática inversa e direta programadas, é possível colocar o código em teste, inicialmente foram realizados testes apenas numéricos, e em seguida testes utilizando recursos visuais.

A função de transformada direta realiza a passagem de uma configuração de juntas para uma posição no espaço cartesiano, e a transformada inversa realiza a passagem contrária, de um ponto no espaço para as configurações de juntas possíveis.

Logo, em teoria, ao utilizar a resposta de uma transformação direta como entrada para uma transformação inversa, o resultado obtido deve ser igual ao parâmetro utilizado como entrada no início do processo. O trecho de código abaixo demonstra o processo utilizado para realizar a “prova real” entre as duas transformações.

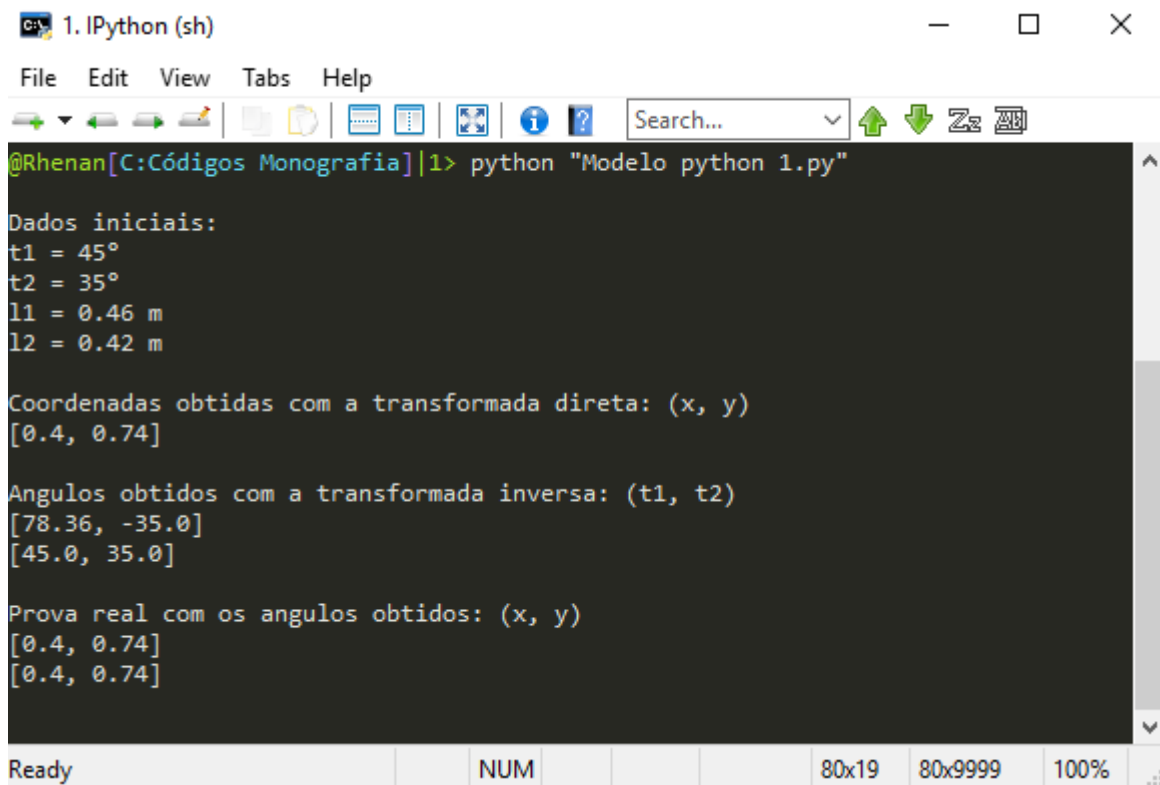
```

1. #Parâmetros de teste
2.
3. #Conjunto de ângulos iniciais
4. t1 = 45
5. t2 = 35
6.
7. #Comprimento dos elos l1 e l2
8. l1 = 0.460
9. l2 = 0.400
10.
11. #Computa transformação direta com ângulos iniciais
12. direct = direct_transform(t1, t2, l1, l2)
13.
14. #Computa transformação inversa partindo das coordenadas obtidas
15. inverse = inverse_transform(direct[2][0], direct[2][1], l1, l2)
16.
17. #Gera prova real para os ângulos obtidos
18. direct1 = direct_transform(inverse[0], inverse[1], l1, l2)
19. direct2 = direct_transform(inverse[2], inverse[3], l1, l2)

```

Os valores utilizados como ângulos iniciais em  $t1$  e  $t2$  são números arbitrários, e podem ser qualquer valor dentro da extensão possível das juntas utilizadas. Para testes, foram consideradas juntas com alcance de  $0^\circ$  a  $360^\circ$ . Os comprimentos dos elos  $l1$  e  $l2$  foram igualmente atribuídos de forma arbitrária, representando elos com comprimentos de 46 cm e 42 cm.

Executando a rotina de testes, os resultados obtidos são visualizados na Figura 29, que mostra o *console* em Python após a execução do *script*. Durante os testes com o modelo programado, foi utilizado como saída para os resultados números o console IPython.



```
1. IPython (sh)
File Edit View Tabs Help
@Rhenan[C:\Códigos Monografia]|1> python "Modelo python 1.py"

Dados iniciais:
t1 = 45°
t2 = 35°
l1 = 0.46 m
l2 = 0.42 m

Coordenadas obtidas com a transformada direta: (x, y)
[0.4, 0.74]

Ângulos obtidos com a transformada inversa: (t1, t2)
[78.36, -35.0]
[45.0, 35.0]

Prova real com os ângulos obtidos: (x, y)
[0.4, 0.74]
[0.4, 0.74]
```

**Figura 29:** Console IPython com resultados de saída do modelo programado.

Para comprovar a veracidade do modelo desenvolvido, serão analisados os três conjuntos de números apresentados como saída:

- **Coordenadas obtidas com a transformada direta:** coordenadas obtidas com a transformação direta, realizada com os dados iniciais. Valores apresentados em coordenadas cartesianas  $x$  e  $y$ ;
- **Ângulos obtidos com a transformada inversa:** ângulos obtidos com a transformação inversa, utilizando as coordenadas anteriores como entrada. Valores apresentados em graus;
- **Prova real com os ângulos obtidos:** os ângulos calculados serão utilizados para retornar para o espaço cartesiano, verificando se o modelo está correto ou não. Valores apresentados em coordenadas cartesianas  $x$  e  $y$ ;

Os passos realizados pela rotina de testes, com os resultados visualizados na saída do console IPython se dão da seguinte maneira:

1. Com os parâmetros iniciais é realizada a transformação direta, onde são calculadas as coordenadas de onde estará a extremidade do manipulador. São apresentadas como saída, as coordenadas finais  $x$  e  $y$ ;
2. Após obter as coordenadas  $x$  e  $y$ , elas são utilizadas como parâmetro para realização da transformação inversa, nesta etapa, são apresentados dois conjuntos de ângulos para  $t1$  e  $t2$ , lembrando que cada ponto no espaço pode ser acessado de duas maneiras diferentes. Observa-se, que um dos conjuntos calculados, é o mesmo utilizado como entrada, mostrando que o modelo segue realizando os cálculos de modo correto.
3. Com o par de ângulos possíveis sendo conhecidos, é realizada a transformação inversa com esses valores. Com um modelo programado corretamente, era de se esperar que as coordenadas apresentadas fossem iguais as obtidas inicialmente, foi possível constatar essa igualdade como conferida no terceiro conjunto de saídas da rotina.

### **3.2.4. VISUALIZAÇÃO DA ÁREA DE TRABALHO**

Comprovando numericamente a veracidade do modelo programado, a próxima etapa é comprovar visualmente o funcionamento do modelo, para isso, foi utilizada a biblioteca *Matplotlib* para primeiro *plotar* a área de trabalho do robô, e depois visualizar posições do robô conforme os valores inseridos de juntas e coordenadas.

Para gerar a área de trabalho, ou *workspace* do manipulador, foi utilizado um método em que é realizada a transformação direta para cada uma das configurações possíveis de ângulos nas juntas  $t1$  e  $t2$ , então as coordenadas do ponto obtido foram armazenadas em vetores para os eixos  $x$  e  $y$ , e por fim, os eixos foram plotados em um gráfico de coordenadas cartesianas.



A primeira etapa, conforme apresentada no código abaixo é definir os parâmetros do manipulador para gerar o *workspace*, definindo tamanho dos elos e ângulos máximo e mínimo para as juntas, bem como os vetores para armazenar as coordenadas dos pontos x e y. Para o modelo em teste, foi utilizado uma extensão de  $-25^\circ$  a  $205^\circ$  para a primeira junta rotacional, e  $-115^\circ$  a  $115^\circ$  para a segunda junta rotacional.

```
1. #Array para armazenar os pontos, em x e y
2. pontos_x = []
3. pontos_y = []
4.
5. #Comprimento dos elos l1 e l2
6. l1 = 0.460
7. l2 = 0.350
8.
9. #Valores máximo e mínimo para ângulo na junta 1
10. t1_min = -25
11. t1_max = 205
12.
13. #Valores máximo e mínimo para ângulo na junta 2
14. t2_min = -115
15. t2_max = 115
```

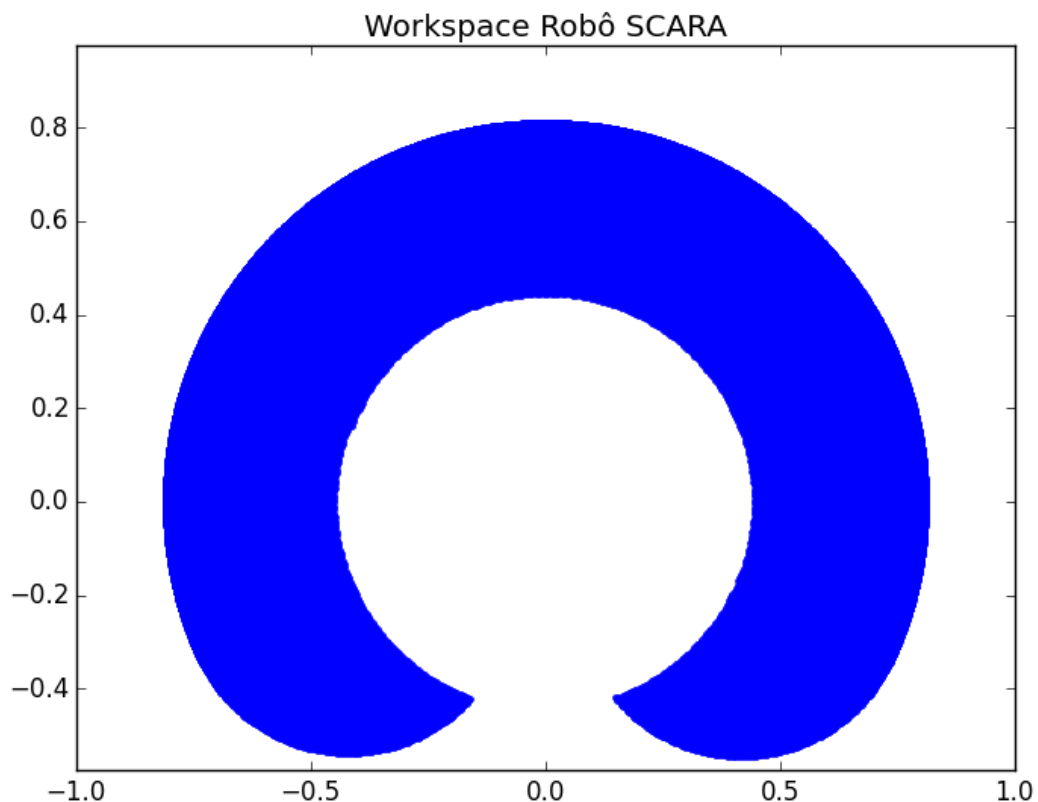
Em seguida, são gerados os pontos em coordenadas cartesianas para cada uma das combinações possíveis dentro do alcance das juntas. Para isso, foi utilizado dois *loops* aninhados que percorrem cada uma dessas combinações, realiza a transformação direta e armazena as coordenadas de resposta nos vetores definidos. Este processo é apresentado no código abaixo.

```
1. #Gera Workspace
2. for i in range(t1_min, t1_max):
3.     for j in range(t2_min, t2_max):
4.
5.         #Computa transformação direta
6.         direct = direct_transform(i, j, l1, l2)
7.
8.         #Adiciona o ponto ao vetor de coordenadas
9.         pontos_x.append(direct[2][0])
10.        pontos_y.append(direct[2][1])
```

Por fim, é necessário visualizar graficamente, realizando a plotagem de um gráfico em um eixo de coordenadas cartesianas com os pontos armazenados nos vetores de posição em  $x$  e  $y$ . A seguir, é apresentado o código que realiza a plotagem do gráfico.

```
1. #Plota Workspace
2. plt.figure()
3. plt.title(u"Workspace Robô SCARA")
4. plt.axis('equal')
5. plt.plot(pontos_x, pontos_y, marker='.', linestyle='none')
6. plt.show()
```

Foi utilizada a função de plotagem da biblioteca Matplotlib, obtendo o gráfico apresentado na Figura 30 a seguir, que representa a área de trabalho do manipulador SCARA, onde os valores representados nos eixos estão em unidade de metros.



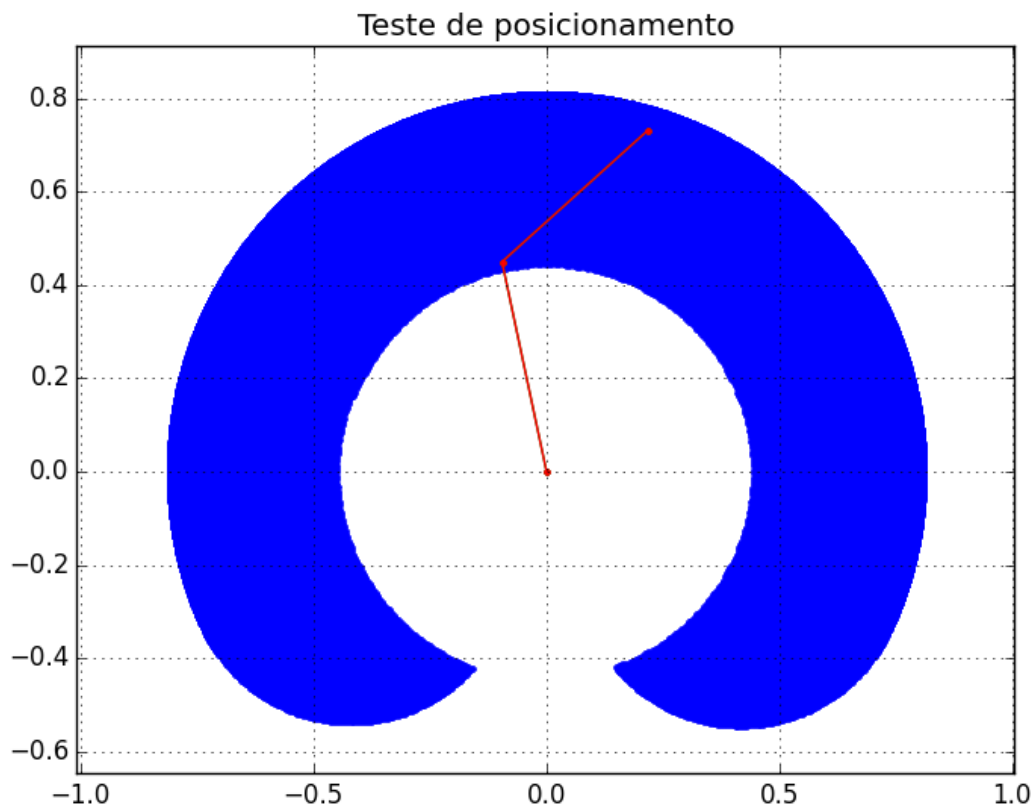
**Figura 30:** Workspace do robô SCARA gerado com o modelo programado.

### 3.2.5. TESTE DE POSICIONAMENTO

Ao realizar os testes para verificar o funcionamento do modelo matemático, todas as saídas eram apenas visualizadas numericamente, a etapa seguinte foi desenvolver um método de visualizar graficamente os resultados apresentados.

Para esta tarefa, foram utilizados os recursos gráficos da linguagem Python, utilizando da biblioteca *Matplotlib*, conforme apresentada na apresentação do *workspace* do robô SCARA.

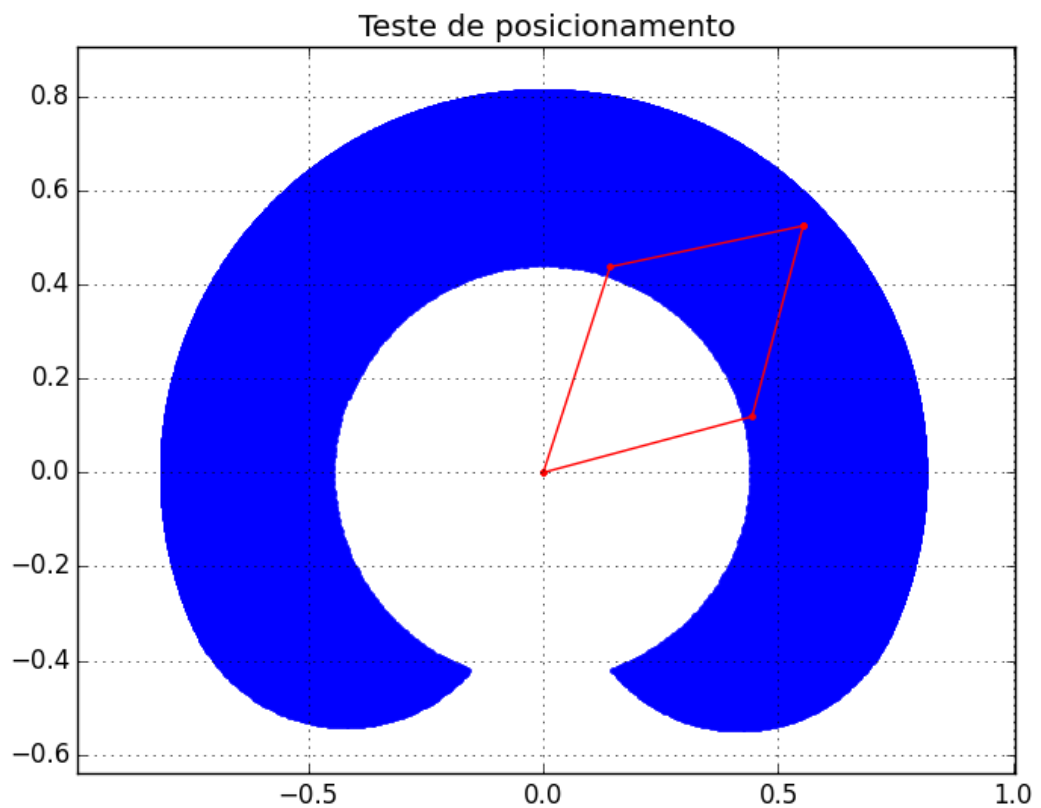
Conhecendo os vetores que representam os elos  $I_1$  e  $I_2$ , é fácil demonstrá-los graficamente, basta traçar uma linha que represente o vetor. A origem para o vetor do elo 1, é o ponto cartesiano  $(0, 0)$ , já a origem para o vetor do elo 2, é o fim do elo 1. A Figura 31 apresenta um exemplo de teste de posicionamento do modelo desenvolvido em Python.



**Figura 31:** Teste de posicionamento com o modelo programado.

Com o teste de posicionamento, é possível visualizar a configuração dos elos do braço robótico, bem como a posição final do atuador e comparar o resultado visto graficamente com os apresentados numericamente. É possível também, demonstrar a característica do manipulador SCARA de oferecer duas configurações possíveis de acesso ao mesmo ponto no espaço.

Gerando dois conjuntos de vetores para os elos, uma para cada configuração possível, pode-se realizar a plotagem dos dois simultaneamente e verificar se ambos alcançam de fato o mesmo ponto no espaço. O teste realizado é observado Figura 32 abaixo.



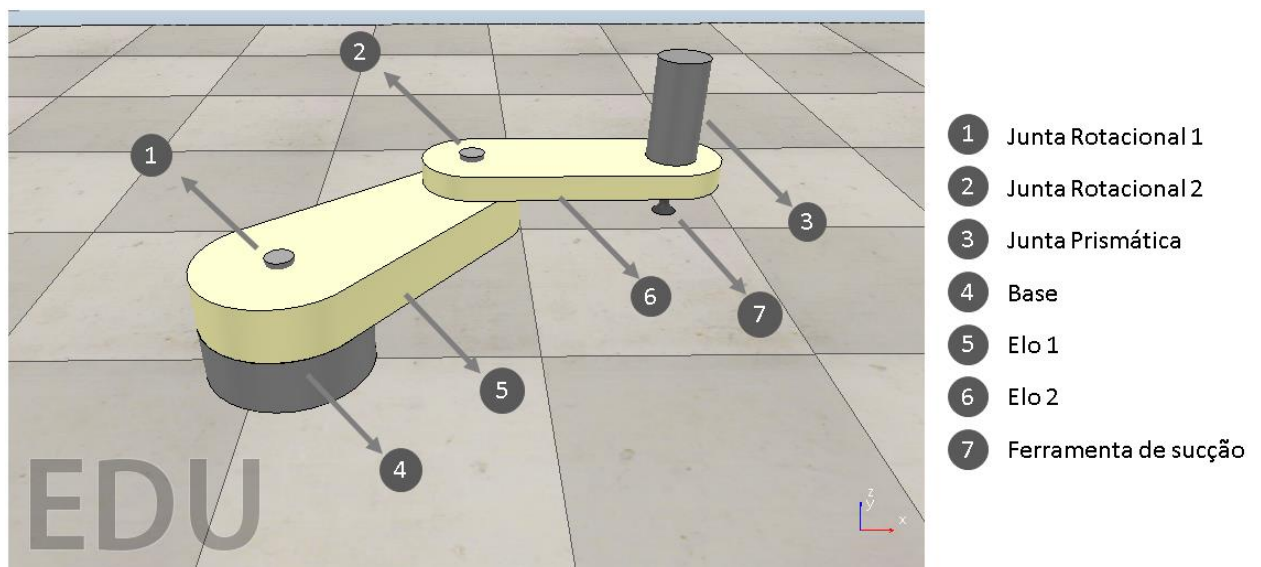
**Figura 32:** Teste de posicionamento com dois acessos a um ponto no espaço.

Cada um dos conjuntos de linhas vermelhas representa uma configuração possível de acesso ao ponto final dentro do *workspace* do robô SCARA. Com o teste de posicionamento foi possível verificar graficamente a veracidade do modelo desenvolvido, a próxima etapa é aplicar o modelo ao *hardware* embarcado no Arduino.

### 3.4. MODELO TRIDIMENSIONAL

A simulação foi realizada com um modelo tridimensional previamente instalado com o *software* V-Rep, provido pela *Coppelia Robotics*. O modelo conta com todos os elementos necessários para simular um modelo simples, mas completamente funcional de manipulador SCARA.

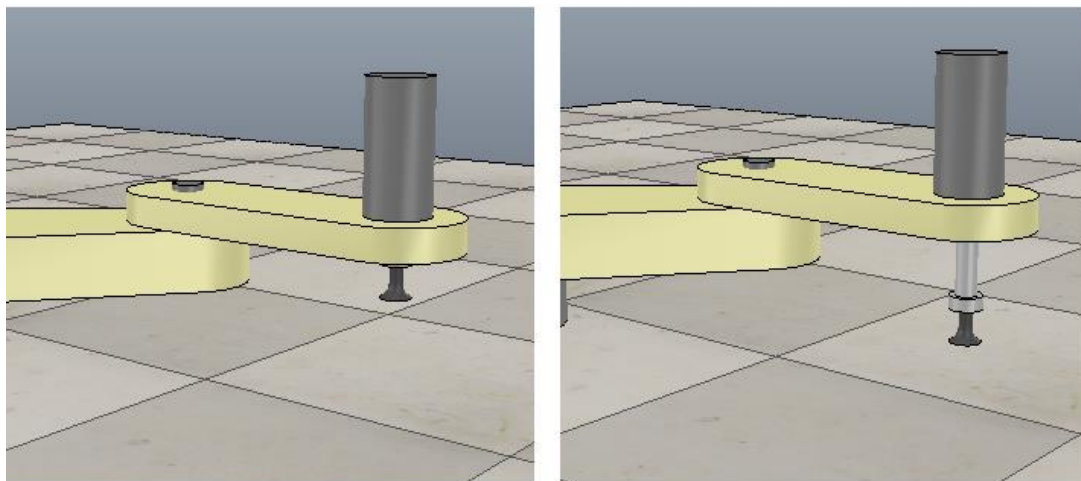
O SCARA utilizado conta com uma base e dois elos rígidos, duas juntas rotacionais, uma junta prismática e uma ferramenta de sucção, conforme observado na Figura 33.



**Figura 33:** Modelo tridimensional SCARA e seus elementos.

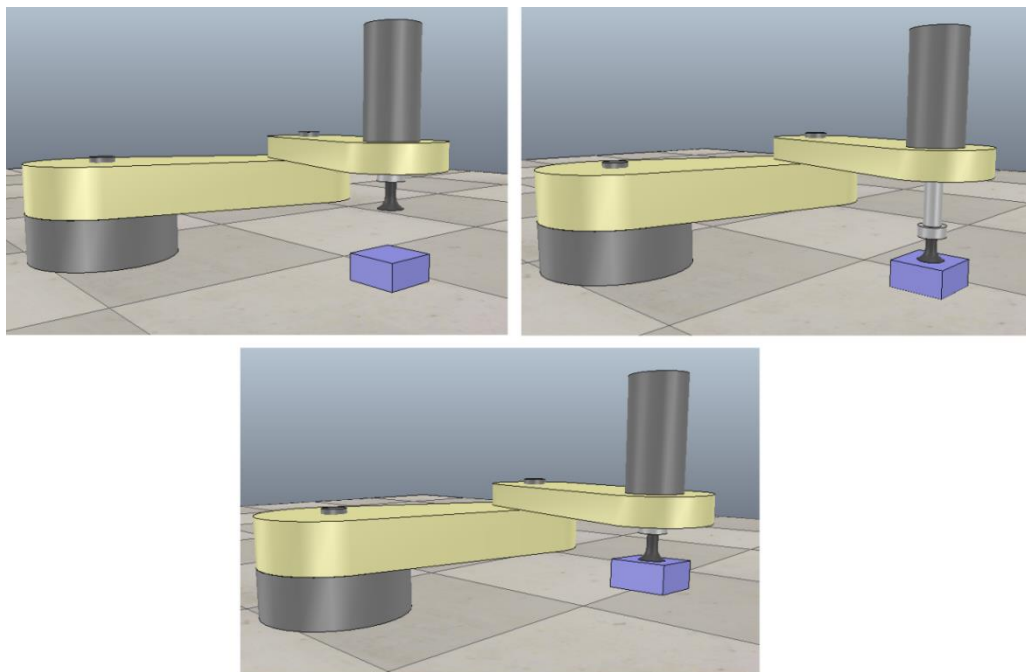
Para o robô realizar movimentos, são escritos valores dos ângulos desejados nas juntas rotacionais 1 e 2, assim é possível localizar a extremidade do robô onde desejado, e para estender ou retrainr o pistão que contém a ferramenta de sucção, são escritos valores na junta prismática, para movimento linear.

A Figura 34 abaixo apresenta a junta prismática em diferentes posições, é possível observar a possibilidade de posicionamento da extremidade do robô. O pistão é movimentado representando o grau de liberdade no eixo z.



**Figura 34:** Acionamento da junta prismática no modelo tridimensional.

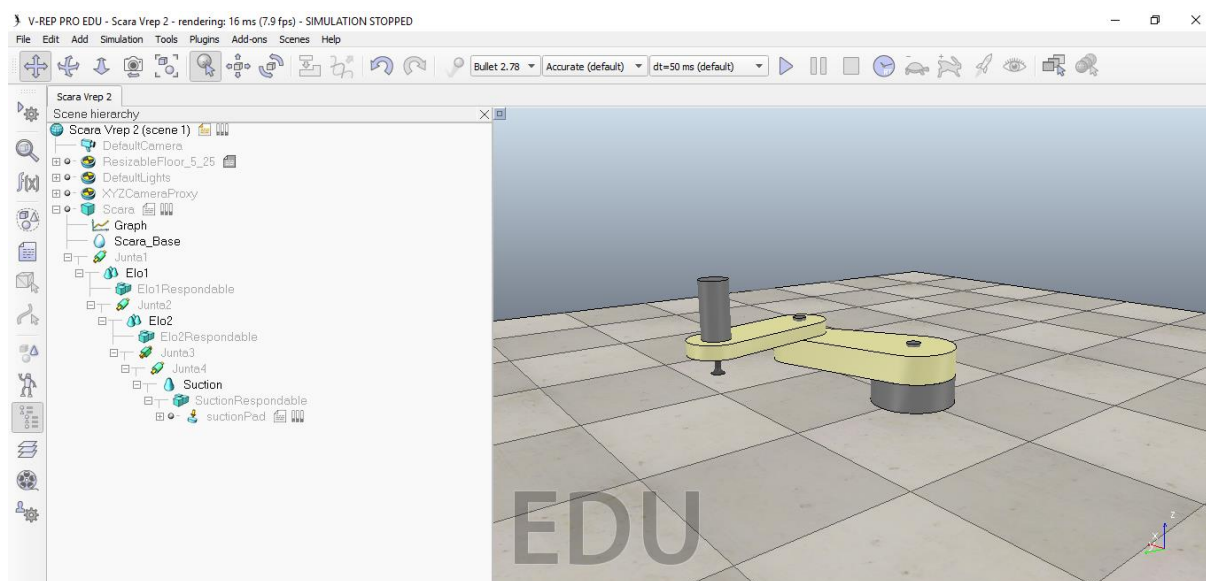
Uma vez que o *software* executa simulações utilizando um motor de física, é possível fazer o robô interagir com outros elementos durante a simulação. Esta característica permite a simulação do funcionamento da ferramenta de sucção utilizada no modelo tridimensional. A Figura 35 a seguir demonstra o funcionamento da ferramenta de sucção.



**Figura 35:** Funcionamento da ferramenta de sucção do modelo tridimensional.

A estrutura do modelo tridimensional no *software* de simulação é feita com base em elementos individuais do robô. Cada parte ou componente do robô é inserido no *software* como um elemento a parte.

Cada elo rígido deve ter um elemento correspondente para o modelo tridimensional e um elemento correspondente para a interação e simulação do *software*. Esses segundos elementos são caracterizados pelo título *respondable* na árvore de elementos do *software*.



**Figura 36:** Visão geral do modelo tridimensional no V-Rep.

Na Figura 36 acima é apresentada uma visão geral do *software* V-Rep. A árvore de elementos presente na cena simulada é apresentada a esquerda. O *software* possui ainda abas para opções de configurações e ferramentas de posicionamento.

Na Figura 37 abaixo, é possível observar com clareza a árvore de elementos da cena simulada, onde o robô utilizado é representado pelo conjunto de elementos nomeado de *Scara*. É possível ainda observar elementos comuns a simulação do V-Rep, como o corpo para representar o solo, iluminação e câmera de visualização do ambiente simulado.



**Figura 37:** Árvore de elementos do modelo tridimensional no V-Rep.

Os elementos rígidos do robô são a base (*Scara\_Base*), os dois elos (*Elo 1*, *Elo 2*), o elemento do pistão e a ferramenta de sucção (*Suction* e *suctionPad*). Cada elemento rígido possui seu respectivo elemento *respondable* para a simulação do modelo.

Os elementos responsáveis pelas juntas são representados pelos nomes de *Junta1* até *Junta4*. As juntas 1 e 2 são respectivamente as responsáveis pelo movimento dos elos 1 e 2, enquanto que a junta 3 é responsável pelo movimento linear do pistão. A junta 4 foi adicionada para realizar a rotação na ferramenta de sucção.



### 3.5. MODELAGEM EM ARDUINO

A implementação do modelo matemático no *Arduino* não é diferente da implementação realizada em linguagem Python, trata-se apenas de uma portabilidade do código escrito em Python para a linguagem do Arduino. O conceito é o mesmo, de aplicar as funções de cinemática inversa e direta, de acordo com os parâmetros fornecidos.

Para realizar a implementação no Arduino, é necessário definir alguns tipos especiais para tratamento de vetores. Foram criados tipos especiais com as funções *typedef* e *struct* para lidar com os vetores representando pontos no espaço e configurações de juntas.

```
1. //Vetor de Posição
2. //Armazena ponto de coordenadas no espaço (x, y)
3. typedef struct
4. {
5.     float x = 0, y = 0;
6. } vector_position;
7.
8. //Vetor de Ângulos
9. //Armazena configuração de ângulos nas juntas (t1, t2)
10. typedef struct
11. {
12.     float t1 = 0, t2 = 0;
13. } vector_angle;
14.
15. //Vetor de Ângulos - Dois pares de ângulos
16. //Armazena ângulos nas juntas em dois pares (t1, t2) e (t3, t4)
17. typedef struct
18. {
19.     float t1 = 0, t2 = 0, t3 = 0, t4 = 0;
20. } vector4_angle;
```

Deste modo, “*vector\_position*” representa um vetor de um ponto no espaço em coordenadas cartesianas ( $x$ ,  $y$ ), enquanto “*vector\_angle*” e “*vector4\_angle*” representam respectivamente configurações de juntas em um par ( $t1$ ,  $t2$ ) e dois pares ( $t1$ ,  $t2$ ) e ( $t3$ ,  $t4$ ). O tipo *vector4\_angle* é utilizado nas situações citadas anteriormente em que um ponto no espaço é acessível através de duas configurações de juntas diferentes.

### 3.4.1. CINEMÁTICA DIRETA EM ARDUINO

Utilizando as mesmas fórmulas já estabelecidas e utilizadas na implementação em Python, é possível realizar a portabilidade do código para Arduino, lidando da mesma maneira com os parâmetros fornecidos.

```
1. vector_position dir_transform(float t1, float t2, float l1, float l2)
2. {
3.     //Define vetores de posição
4.     vector_position r1, r2, r3;
5.
6.     //Calcula vetores individuais para Elo 1
7.     r1.x = l1 * cos(radians(t1));
8.     r1.y = l1 * sin(radians(t1));
9.
10.    //Calcula vetores individuais para Elo 2
11.    r2.x = l2 * cos(radians(t1 + t2));
12.    r2.y = l2 * sin(radians(t1 + t2));
13.
14.    //Realiza soma vetorial para posição final
15.    r3.x = r1.x + r2.x;
16.    r3.y = r1.y + r2.y;
17.
18.    //Retorna vetor com coordenadas finais
19.    return r3;
20. }
```

A cinemática direta realiza a passagem do espaço das juntas para o espaço cartesiano, deste modo, semelhantemente a programação realizada em Python, os parâmetros recebidos são os comprimentos dos elos  $l1$  e  $l2$ , e os ângulos da configuração de juntas  $t1$  e  $t2$ . A resposta é um vetor de coordenadas com a posição da extremidade para esta configuração de parâmetros.

O procedimento realizado é o mesmo do modelo matemático e em Python, primeiro são calculados a posição para vetores individuais dos elos  $l1$  e  $l2$ , e em seguida é realizada uma operação de soma vetorial para definir a posição final do manipulador.

### 3.4.2. CINEMÁTICA INVERSA EM ARDUINO

Novamente de modo semelhante ao implementado em Python, o código é escrito de maneira a funcionar na plataforma Arduino, mantendo as equações utilizadas e a mesma configuração de parâmetros.

```
1. vector4_angle inv_transform(float x, float y, float l1, float l2)
2. {
3.     //Define vetor de resposta e variáveis para cálculo
4.     vector4_angle res;
5.     float t1, t2, num, den;
6.
7.     //Calcula t2 para situação com t2 positivo
8.     t2 = acos((pow(x,2)+pow(y,2)-pow(l1,2)-pow(l2,2))/(2*l1*l2));
9.
10.    //Verifica exceção na função acos
11.    if (isnan(t2)) t2 = 0;
12.
13.    //Calcula t1 para situação com t2 positivo
14.    num = y * (l1 + l2 * cos(t2)) - x * l2 * sin(t2);
15.    den = x * (l1 + l2 * cos(t2)) + y * l2 * sin(t2);
16.    t1 = atan2(num, den);
17.
18.    //Realiza conversão de radianos para graus
19.    t1 = degrees(t1);
20.    t2 = degrees(t2);
21.
22.    //Adiciona primeiro conjunto de ângulos ao vetor final
23.    res.t1 = t1;
24.    res.t2 = t2;
25.
26.    //Calcula t2 para situação com t2 negativo
27.    t2 = -acos((pow(x,2)+pow(y,2)-pow(l1,2)-pow(l2,2))/(2*l1*l2));
28.
29.    // Verifica exceção na função acos
30.    if (isnan(t2)) t2 = 0;
31.
32.    //Calcula t1 para situação com t2 negativo
33.    num = y * (l1 + l2 * cos(t2)) - x * l2 * sin(t2);
34.    den = x * (l1 + l2 * cos(t2)) + y * l2 * sin(t2);
35.    t1 = atan2(num, den);
36.
37.    //Realiza conversão de radianos para graus
38.    t1 = degrees(t1);
39.    t2 = degrees(t2);
40.
41.    //Adiciona segundo conjunto de ângulos ao vetor final
42.    res.t3 = t1;
43.    res.t4 = t2;
44.
45.    //Retorna vetor final com dois pares de ângulos
46.    return res;
47. }
```

São recebidos como parâmetros os comprimentos dos elos  $l_1$  e  $l_2$ , e as coordenadas do ponto no espaço  $x$  e  $y$ , e então é realizada a passagem do espaço cartesiano para o espaço das juntas. A resposta retornada é um vetor de dois conjuntos de ângulos,  $t_1$  e  $t_2$  para a primeira possibilidade de acesso ao ponto e  $t_3$  e  $t_4$  para a segunda possibilidade de acesso ao ponto.

Nota-se uma rigidez maior no desenvolvimento da portabilidade do código escrito em Python para o código escrito em Arduino, principalmente em relação ao tratamento de vetores, onde por consequência o código implementado em Arduino se torna maior em quantidade de linhas.

Tendo sido implementadas as funções de cinemática direta e inversa na plataforma embarcada, a próxima etapa é prover comunicação entre o hardware e o *software* de simulação para dar início ao desenvolvimento das funções de movimento do robô.

### 3.6. INTERFACES DE COMUNICAÇÃO

Para a continuidade do projeto, é necessário realizar a comunicação entre *hardware* e *software* de simulação, para isso é necessário desenvolver duas interfaces de comunicação, uma entre o Arduino e o *client* em Python, e outra entre o *client* e o *software* V-Rep.

Ambas as interfaces de comunicação devem ser capazes de enviar e receber informações. Todas as informações são transmitidas em forma de números, assim as interfaces devem ser capazes de transmitir e receber números tratando a precisão decimal adequada para cada situação.

#### 3.6.1. INTERFACE ARDUINO – PYTHON

Para a comunicação entre o hardware (Arduino) e o *client* escrito em Python, foi utilizado o protocolo de comunicação Serial, através de uma porta COM (USB). Esta funcionalidade é empregada na linguagem Python através da biblioteca *serial*, e por padrão está presente na plataforma Arduino.

Antes de iniciar o processo de comunicação é necessário se atentar a alguns detalhes importantes. Por se tratar de uma comunicação entre duas linguagens diferentes, é preciso um cuidado especial ao tratar dados nas duas plataformas, assim é preciso transmitir dados numéricos com a mesma precisão decimal em ambas as direções de comunicação.

Um número com precisão decimal é representado no Arduino pelo tipo de dado *float*, que é definido por um dado de 4 *bytes*. Já um número do tipo inteiro é representado, por um dado de 2 *bytes*. A Tabela 5 abaixo apresenta os tipos de dados e seus limites para dois tipos, inteiros e de ponto flutuante.

**Tabela 5:** Tipos de dados *int* e *float* para a plataforma Arduino.

Tipo de dado	Tamanho	Amplitude
int	2 Bytes	-32767 a 32767
float	4 Bytes	-3.4E+38 a 3.4E+38

Tendo isso em mente, é necessário lembrar que a quantidade de dados a serem transmitidos impacta diretamente na velocidade da transmissão. Assim, uma vez que um número do tipo *float*, consome mais memória, é preferível então, que os dados sejam tratados como tipo inteiro.

Para o desenvolvimento do projeto, foi adotada precisão decimal de  $0,1^\circ$  para tratamento dos ângulos. Assim, considerando uma junta rotacional de 0 a  $360^\circ$ , os números tratados pelo modelo são de  $-360,0^\circ$  a  $360,0^\circ$ . Números negativos são considerados para indicar movimentos na direção contrária à do ponto  $0^\circ$ .

Durante o processamento pelo hardware e pelo *software* de simulação, os números são tratados de maneira normal, com ponte flutuante e precisão decimal adotada, é realizada então uma adaptação apenas para transmitir o dado numérico através da porta serial.

Essa adaptação é realizada através de uma operação de multiplicação, que computacionalmente apresenta impacto mínimo na velocidade de processamento comparado ao para transmitir um número de ponto flutuante de 4 *bytes*.

A Tabela 6 apresenta alguns exemplos de conversão para o processo de transmissão de valores que são enviados ao *client*, que por sua vez, irá repassar o valor ao V-Rep.

**Tabela 6:** Conversão de dados para o processo de transmissão.

Ângulo (4 bytes)		Adaptação para inteiro		Valor Transmitido (2 bytes)
0,0°	→	$\times 10$	→	0
45,0°	→	$\times 10$	→	450
90,0°	→	$\times 10$	→	900
95,4°	→	$\times 10$	→	954
360,0°	→	$\times 10$	→	3600

Semelhantemente, é realizada uma conversão nos dados recebidos pelo Arduino vindos do client, alguns exemplos são apresentados na Tabela 7 abaixo.

**Tabela 7:** Conversão de dados para o processo de recepção.

Valor Recebido (2 bytes)		Adaptação para ponto flutuante		Valor Utilizado (4 bytes)
0	→	÷ 10	→	0,0°
450	→	÷ 10	→	45,0°
900	→	÷ 10	→	90,0°
95,4	→	÷ 10	→	95,4°
3600	→	÷ 10	→	360,0°

A próxima etapa é realizar a comunicação de fato. Antes de iniciar uma transmissão através do protocolo serial, é necessário se conectar a uma porta COM, o trecho de código abaixo escrito em Python realiza a conexão entre a biblioteca serial e a plataforma Arduino.

```

1. #Executa conexão com porta COM
2. def connect_com(com_port, baud_rate):
3.     #Tenta conexão com porta COM
4.     try:
5.         comport = serial.Serial(com_port, baud_rate)
6.     except:
7.         print "Falha ao conectar porta COM.\n"
8.         sys.exit(0)
9.
10.    print u"Conexão estabelecida.\n"
11.
12.    return comport

```

A função de conexão com a porta COM recebe dois parâmetros, um deles é a porta em questão a ser conectada, no caso a mesma que se encontra a plataforma Arduino, e o outro parâmetro é o *baud rate* da conexão, que nada mais é do que a velocidade de transmissão de dados.

Para realizar a transmissão entre Python e Arduino colocando em prática o método de conversão de ponto flutuante para valores inteiros, foi preciso antes definir uma função para arredondamento de valores com a precisão decimal desejada.

Por padrão, um número do tipo ponto flutuante em Arduino tem precisão de 6 casas decimais, no decorrer do projeto foi utilizado apenas uma casa de precisão decimal. O código abaixo, escrito em Arduino, realiza a adaptação de um número de ponto flutuante para x casas decimais.

```
1. //Arredonda valor em X casas decimais
2. float roundx(float value, byte dec)
3. {
4.     return round(value * int(pow(10, dec))) / pow(10, dec);
5. }
```

Deste modo, é possível exemplos das funções utilizadas para leitura e escrita de valores via porta serial no Arduino. O código abaixo, escrito em Arduino, apresenta uma exemplificação do processo de escrita e leitura, incluindo as adaptações demonstradas.

```
1. void escrita(float value)
2. {
3.     //Trata dados a enviar
4.     int send_value = roundx(value, 1) * 10;
5.
6.     //Envia valor via Serial
7.     Serial.write((const char *) & send_value, sizeof(int));
8. }
9.
10. float leitura()
11. {
12.     //Buffer e variável para receber dados via Serial
13.     unsigned char buffer[2];
14.     int incoming_value;
15.
16.     //Executa leitura da porta Serial
17.     Serial.readBytes(buffer, sizeof(int));
18.     memcpy(& incoming_value, buffer, sizeof(int));
19.
20.     //Executa adaptação do valor lido
21.     return roundx(incoming_value / 10.0f, 1);
22. }
```



Nota-se que todo o processo foi feito com funções específicas de modo a possibilitar uma comunicação na maior velocidade possível, não optando pelas funções pré-estabelecidas na linguagem de Arduino para comunicação serial, e implementando as funções adaptadas para o projeto em questão.

O mesmo processo é realizado na linguagem Python para escrita e leitura de dados via serial, incluído o processo de adaptação do valor recebido e enviado para a precisão decimal adequada.

No *client* foi utilizada a biblioteca nativa da linguagem *Struct*, para lidar com os bytes recebidos e enviados via serial, de modo a adapta-los para a transmissão. O código abaixo, escrito em Python, apresenta funções análogas às utilizadas para escrita e leitura.

```
1. def escrita(value):
2.
3.     #Ajusta valores para envio via serial
4.     value = (round(value * 10))
5.
6.     #Realiza pack de valores em inteiros de 2 bytes
7.     send_value = struct.pack('<h', value)
8.
9.     #Executa escrita dos bytes via serial
10.    comport.write(bytes(send_value))
11.
12. def leitura():
13.
14.     #Leitura de valores via serial
15.     incoming_value = comport.read(2)
16.
17.     #Realiza unpack de valores em inteiros de 2 bytes
18.     value_serial = struct.unpack('<h', incoming_value)[0]
19.
20.     #Realiza adaptação dos valores lidos
21.     return value_serial /= 10.0
```

### 3.6.2. INTERFACE PYTHON – V-REP

A interface entre o *client* Python e o software V-Rep, é a responsável por enviar os valores a serem escritos nas juntas do modelo simulado, receber os valores de sensoramento da simulação, e repassar e receber os devidos valores do hardware Arduino.

Toda a comunicação com o software de simulação, é feita através da V-Rep API, com suporte para a linguagem Python, utilizando os comandos disponíveis para realizar a “conversa” com o *software*. A utilização da API é feita realizando a importação da biblioteca *vrep* fornecida pela Coppelia Robotics.

Antes de iniciar qualquer conversa entre o client e o software V-Rep, é necessário realizar a conexão entre a API e o software, como demonstrado no código em Python abaixo.

```
1. import vrep
2.
3. #Realiza conexão com o software V-rep
4. def connectVREP():
5.     #Fecha conexões existentes
6.     vrep.simxFinish(-1)
7.
8.     #Define objeto de conexão ao V-rep
9.     clientID = vrep.simxStart('127.0.0.1', 19997 , True, True, 5000, 5)
10.
11.     #Verifica status da conexão
12.     if clientID != -1:
13.         #Conectado com sucesso ao V-Rep
14.         print "Conectado ao Vrep.\n"
15.
16.         #Define modo de conexão como síncrono
17.         vrep.simxSynchronous(clientID, False)
18.
19.         #Retorna objeto de conexão
20.         return clientID
21.     else:
22.         #Falha na conexão com o V-Rep
23.         print "Erro ao conectar ao Vrep!\n"
24.         sys.exit(0)
```

Os primeiros parâmetros para a função de conexão com o V-Rep são respectivamente o endereço de conexão e a porta de conexão. Em seguida tem-se parâmetros que indicam se o *software* deve aguardar pela resposta de comunicação e se deve tentar se reconectar ao perder o *link* com o V-Rep. Por fim, tem-se o *timeout* da conexão e o intervalo de tempo para enviar e receber novos pacotes, ambos em milissegundos.

Após a conexão, o objeto responsável pela comunicação com o V-Rep é o *objectID*, obtido como retorno da função de conexão. Uma vez realizada a conexão, é possível iniciar a conversa com a simulação.

Uma vez que todos os ângulos das juntas são calculados pelo Arduino, o *software* atua como um sistema supervisor e de sensoriamento. Sendo assim, os ângulos tratados pelo *hardware* devem ser escritos nas juntas do modelo tridimensional.

Para que o client se comunique com as juntas do modelo tridimensional, é necessário ter um objeto de referência para cada uma delas. Nesse processo, é necessário referenciar a junta com o mesmo nome que está presente na árvore do modelo tridimensional, apresentado na Figura 37.

O Código em Python a seguir apresenta a definição de objetos para cada uma das juntas, de “Junta1” a “Junta4”.

```
1. #Define objeto para Junta1 - Rotação do Elo 1
2. ret, joint1_handler =
3.     vrep.simxGetObjectHandle(clientID, "Junta1", vrep.simx_opmode_oneshot_wait)
4.
5. #Define objeto para Junta2 - Rotação do Elo 2
6. ret, joint2_handler =
7.     vrep.simxGetObjectHandle(clientID, "Junta2", vrep.simx_opmode_oneshot_wait)
8.
9. #Define objeto para Junta3 - Movimento em Z
10. ret, joint3_handler =
11.     vrep.simxGetObjectHandle(clientID, "Junta3", vrep.simx_opmode_oneshot_wait)
12.
13. #Define objeto para Junta4 - Rotação da Ferramenta
14. ret, joint4_handler =
15.     vrep.simxGetObjectHandle(clientID, "Junta4", vrep.simx_opmode_oneshot_wait)
```

Deste modo, obtém-se quatro objetos para lidar com as juntas, de “*joint1\_handler*” a “*joint4\_handler*”. Toda a interação com as juntas será feita através destes 4 *handlers*.

Após definir os objetos para cada junta, é possível comunicar-se diretamente com elas. O processo é realizado através de funções de escritas de valores, para atuação, e leitura dos valores das juntas, para sensoriamento.

O código abaixo, apresenta duas funções de escrita para atuação. A primeira para juntas rotativas, onde o valor escrito é um ângulo em graus, e a segunda para juntas prismáticas, onde o valor escrito é uma posição, em metros.

```
1. #Escreve ângulo de posição da Junta
2. def write_angle(clientID, handler, ang):
3.     vrep.simxSetJointPosition(clientID,handler,radians(ang),vrep.simx_opmode_oneshot)
4.
5. #Escreve posição na junta
6. def write_pos(clientID, handler, m):
7.     vrep.simxSetJointPosition(clientID,handler, m,vrep.simx_opmode_oneshot)
```

Do mesmo modo, são definidas duas funções para leitura de valores das juntas durante o sensoriamento, uma função para leitura de juntas rotativas, em ângulos, e outra para leitura de juntas prismáticas, em posição. As funções são apresentadas no código em Python abaixo.

```
1. #Executa leitura de ângulo da junta
2. def read_angle(clientID, handler):
3.     joint = vrep.simxGetJointPosition(clientID,handler,vrep.simx_opmode_oneshot_wait)
4.     joint = round(degrees(joint), 1)
5.     return joint
6.
7. #Executa leitura de posição da junta
8. def read_position(clientID, handler):
9.     joint = vrep.simxGetJointPosition(clientID,handler,vrep.simx_opmode_oneshot_wait)
10.    return joint
```

Assim, é possível escrever e ler valores de juntas no modelo tridimensional durante a simulação, realizando ações de posicionamento e de sensoriamento, e integrando com a interface de comunicação entre o Arduino, a integração *hardware-software* se faz completa.

### 3.7. FUNÇÕES DE MOVIMENTO

Seguindo a estrutura de um sistema desenvolvido com tecnologia *Hardware in the Loop*, o objetivo do projeto é simular em um ambiente controlado uma situação mais próxima do real possível, sendo assim, foi desenvolvida uma linguagem de programação para construção das rotinas de movimentação inspirada em linguagens industriais reais.

Foi utilizada como parâmetro de inspiração, a linguagem de robótica *RAPID*, desenvolvida pela ABB, adaptando a sintaxe geral do sistema para a linguagem C/C++ do Arduino. No total, foram desenvolvidas sete funções de movimentos disponíveis para utilização na programação de rotinas.

Uma vez que é possível localizar o braço mecânico em um ponto do espaço, utilizando as funções de cinemática inversa e direta, o procedimento para construir trajetórias é juntar uma série de pontos no espaço de modo a realizar o movimento desejado.

Para programar a movimentação no plano cartesiano do robô, foram desenvolvidas basicamente três funções de movimento, que tem seus escopos apresentadas no código abaixo, responsável por realizar movimento dos graus de liberdades dos elos, posicionando a extremidade do robô em um determinado ponto.

```
1. //Movimenta para posição no espaço das juntas
2. void moveJ(float t1, float t2, bool interpolation);
3.
4. //Movimenta para posição no espaço cartesiano
5. void moveP(float x, float y, bool interpolation);
6.
7. //Movimento linear em direção a um ponto no espaço cartesiano
8. void moveL(float x, float y, int points);
```

Essas três funções inicialmente são as mais básicas para a movimentação da estrutura do manipulador robótica, sendo desenvolvidas mais duas variantes apresentadas mais adiante.

Cada função de movimento recebe como parâmetros os valores referentes ao seu tipo de movimento, e a descrição de cada uma é a seguinte:

- **moveJ:** Se move para o conjunto de ângulos inseridos como parâmetro, realizando movimento no espaço das juntas, utiliza como parâmetros os ângulos desejados nas juntas rotacionais  $\theta_1$  e  $\theta_2$ .
- **moveP:** Posiciona a extremidade do manipulador em um ponto no espaço cartesiano, utiliza como parâmetro as coordenadas desejadas, realizando movimento no plano cartesiano  $(x, y)$ .
- **moveL:** Realiza movimento linear da posição atual para a posição cartesiana inserida como parâmetro, realizando movimento no plano cartesiano  $(x, y)$ .

Para as funções *moveJ* e *moveP*, o terceiro parâmetro é uma variável *booleana* nomeada de *interpolation*, e diz se o movimento será realizado utilizando interpolação entre as juntas ou não. Enquanto que para a função de movimento linear *moveL*, o terceiro parâmetro é uma variável do tipo inteiro que indica quantos pontos serão compreendidos na reta desenvolvida pelo movimento linear.

A interpolação nessas duas primeiras funções indica se o movimento das juntas será realizado de maneira síncrona, ou seja, se as juntas rotacionais  $\theta_1$  e  $\theta_2$  irão terminar o movimento juntas, ou assíncrona, terminando o movimento em momentos diferentes.

Para a função *moveL*, foi utilizado um algoritmo de interpolação linear, de modo a combinar a quantidade de pontos especificada como parâmetro dentro do intervalo entre as coordenadas  $x$  e  $y$  do ponto desejado, formando assim, uma linha reta entre a posição atual do manipulador, e a desejada.

O comportamento detalhado de cada uma das funções apresentadas será detalhado e demonstrado melhor no capítulo 4 de resultados e discussões, apresentando exemplos e alterações de cada uma delas.

Partindo das três funções básicas de movimento *moveJ*, *moveP* e *moveL*, são derivadas mais duas funções e tem seus respectivos escopos apresentadas no trecho de código a seguir, são as funções que utilizam o conceito de *offset*.

```
1. //Movimento no plano cartesiano a partir de posição de offset
2. void moveP_Offs(vector_position point, float x, float y, bool interpolation)
3.
4. //Movimento linear a partir de posição de offset
5. void moveL_Offs(vector_position point, float x, float y, int points)
```

O conceito de *offset* foi aplicado definindo que o movimento do manipulador robótico irá partir de um ponto de referencia no espaço, e não da origem do robô. Assim, caso seja necessário reposicionar a trajetória do robô, basta modificar a posição de origem, não sendo necessário alterar todas as coordenadas envolvidas no processo.

A estrutura da função é basicamente a mesma das funções sem a função de *offset*, adicionando apenas um parâmetro ao inicio, do tipo definido no inicio da modelagem como *vector\_positon*, que indica um vetor de posição no espaço. Assim, os valores apresentados como parâmetros para coordenadas *x* e *y* serão adicionados (função *offset*) ao valor da posição do vetor informado como referencia.

Para a manipulação da extremidade, foram criadas duas funções para controle da ferramenta no manipulador robótico, uma para realizar o movimento no eixo *z*, e outra para realizar a rotação da ferramenta do robô SCARA, que tem seus escopos apresentadas no trecho de código abaixo.

```
1. //Movimento do eixo Z
2. void moveZ(float z, float speed);
3.
4. //Movimento de rotação da ferramenta
5. void moveR(float ang, float step
```

As duas funções dizem respeito à movimentação da ferramenta do manipulador, e cada uma é descrita da seguinte maneira:

- **moveZ:** Realiza movimento no terceiro grau de liberdade do robô SCARA, posicionando a ferramenta em uma posição no eixo z do *workspace* do manipulador.
- **moveR:** Realiza movimento de rotação na ferramenta do manipulador, representando uma junta rotativa na ferramenta do robô SCARA.

A função *moveZ* recebe como parâmetro dois valores do tipo *float* que indicam a posição no eixo z em que o robô deve posicionar a ferramenta e a velocidade com que será realizada a movimentação.

Já a função *moveR* recebe parâmetros que indicam a rotação em graus a ser aplicada a ferramenta, e uma variável representando a quantidade de graus por movimentação realizada, ou seja, indica a precisão em graus do movimento de rotação.

Por fim, foram criadas também duas funções para realizar o controle da ferramenta empregada nas aplicações, uma ferramenta de ventosa, com sistema de sucção. As funções servem para ativar ou desativar o sistema, e tem seus escopos apresentadas a seguir.

```
1. //Ativa sucção da ventosa
2. void suction_on();
3.
4. //Desativa sucção da ventosa
5. void suction_off();
```

As funções de controle da sucção da ventosa não recebem nenhum parâmetro e funcionam de maneira persistente, ou seja, após ser ativada com a função *suction\_on*, a sucção só será desativada se receber o comando *suction\_off*.



Assim, foram desenvolvidas sete funções de movimento, e duas funções de controle de ativação da ferramenta, e todas as rotinas desenvolvidas para teste do sistema foram feitas utilizando estas funções. O trecho de código abaixo apresenta uma visão geral dos escopos das funções desenvolvidas.

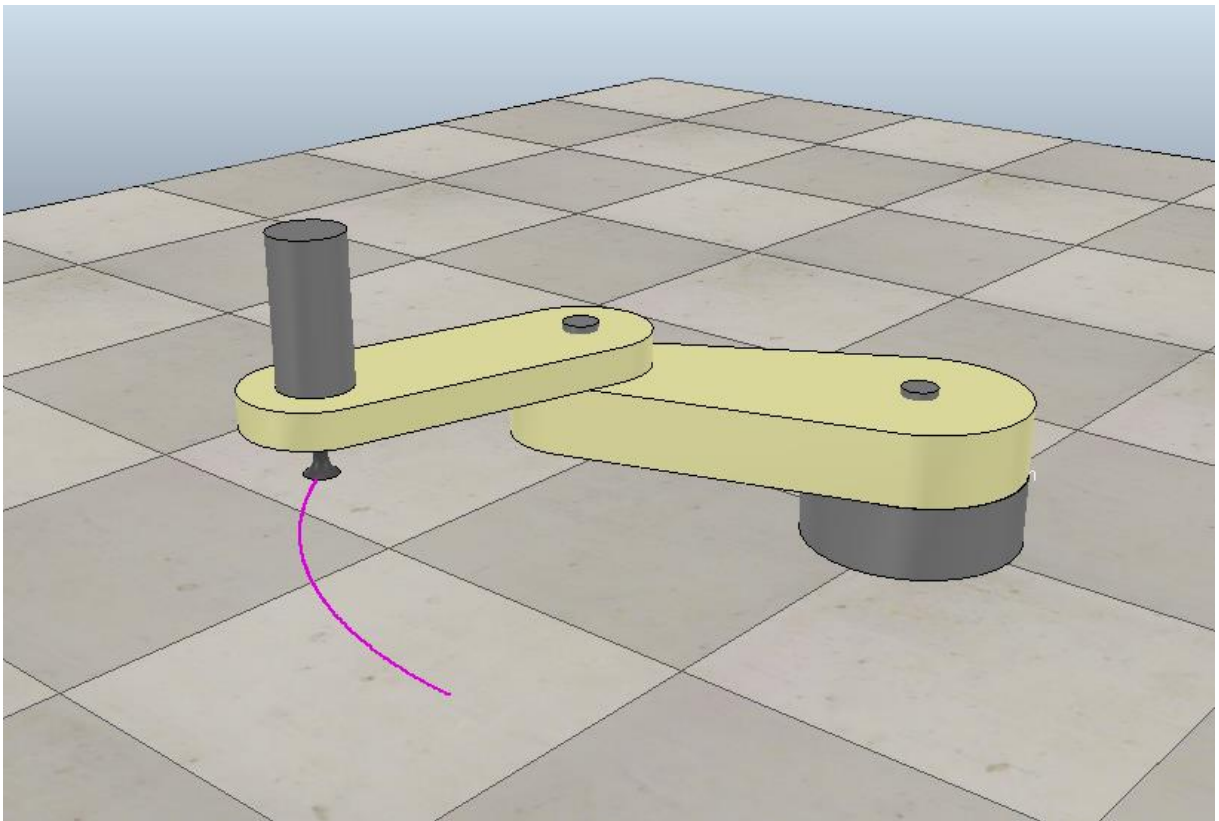
```
1. //Funções básicas de movimentação
2.
3. //Movimenta para posição no espaço das juntas
4. void moveJ(float t1, float t2, bool interpolation);
5.
6. //Movimenta para posição no espaço cartesiano
7. void moveP(float x, float y, bool interpolation);
8.
9. //Movimento linear em direção a um ponto no espaço cartesiano
10. void moveL(float x, float y, int points);
11.
12.
13. //Funções de movimentação com Offset
14.
15. //Movimento no plano cartesiano a partir de posição de offset
16. void moveP_Offs(vector_position point, float x, float y, bool interpolation);
17.
18. //Movimento linear a partir de posição de offset
19. void moveL_Offs(vector_position point, float x, float y, int points);
20.
21.
22. //Funções de movimentação da ferramenta
23.
24. //Movimento do eixo Z
25. void moveZ(float z, float speed);
26.
27. //Movimento de rotação da ferramenta
28. void moveR(float ang, float step);
29.
30.
31. //Funções de ativação da ferramenta
32.
33. //Ativa sucção da ventosa
34. void suction_on();
35.
36. //Desativa sucção da ventosa
37. void suction_off();
```

#### 4. RESULTADOS E DISCUSSÃO

Utilizando as funções de movimento desenvolvidas, foram criadas diversas trajetórias e situações para colocar o modelo matemático e a simulação em prova, incluído testes de construção de trajetórias e posicionamento de objetos.

O procedimento utilizado para verificar a trajetória desenvolvida pelo robô foi utilizar a função de construção de gráficos de trajetórias do *software* V-Rep, com ela é possível armazenar a posição de objetos do modelo tridimensional durante o período de simulação.

Deste modo, este recurso foi utilizado como uma espécie de “caneta” que realizava a marcação da extremidade do robô através do tempo de simulação. A Figura 38 abaixo mostra a função em execução.



**Figura 38:** Demonstração da ferramenta de construção de gráficos do V-Rep.

Para executar o controle da função de gráfico do *software* V-Rep, foram desenvolvidas mais duas funções de comando adicionais, que não representam controle sobre o robô em si, mas sim sob o *software* de simulação, iniciando ou finalizando a coleta de posição da extremidade do manipulador, conforme os escopos apresentados a seguir.

```
1. //Inicia coleta de posições
2. plot_start();
3.
4. //Finaliza coleta de posições
5. plot_end();
```

É possível ainda, exportar as posições contidas no gráfico de trajetória como um arquivo de extensão *.csv*, e assim, realizar a plotagem de gráficos utilizando a biblioteca *Matplotlib* para a linguagem Python.

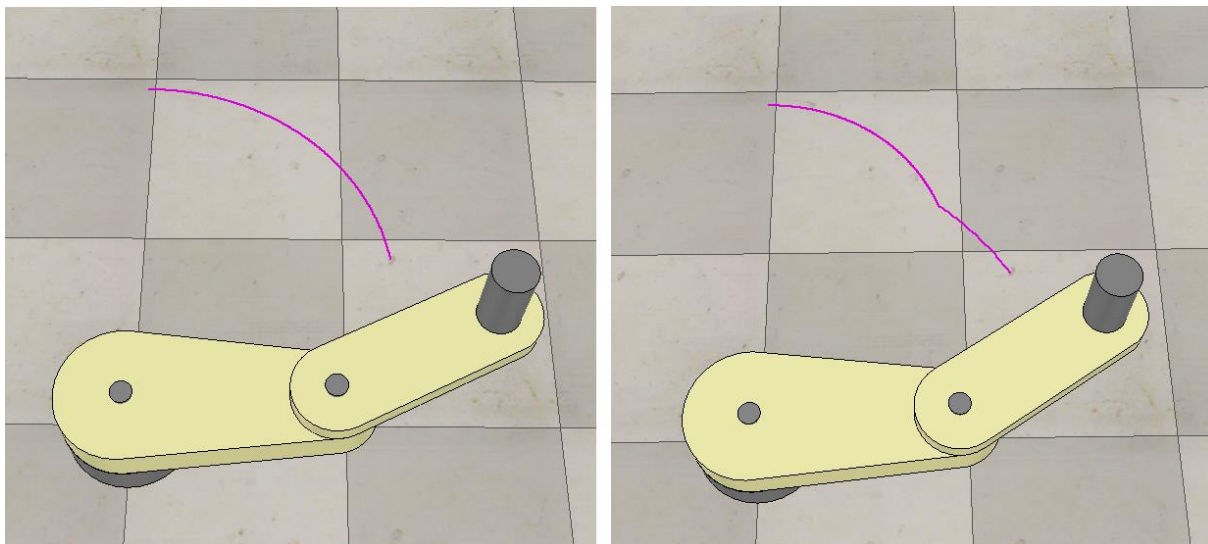
#### 4.1. MOVIMENTO DE JUNTAS

Com o gráfico auxiliando a construção das rotinas, foi realizado o primeiro teste de funcionamento. Utilizando a função de movimento *moveJ*, foram testados o comportamento da função realizando movimento entre dois conjuntos de ângulos para as juntas rotacionais  $\theta_1$  e  $\theta_2$ .

O trecho de código abaixo mostra a rotina utilizada para testar a função *moveJ*. Foram realizados testes alterando o parâmetro de interpolação de juntas para *true* e *false*.

```
1. void teste_movej()
2. {
3.     //Movimenta para conjunto inicial de ângulos
4.     moveJ(90, 0, true);
5.
6.     //Inicia gráfico de trajetória
7.     plot_start();
8.
9.     //Movimenta para conjunto final de ângulos
10.    moveJ(0, 70, true);
11.
12.    //Finaliza gráfico de trajetória
13.    plot_end();
14. }
```

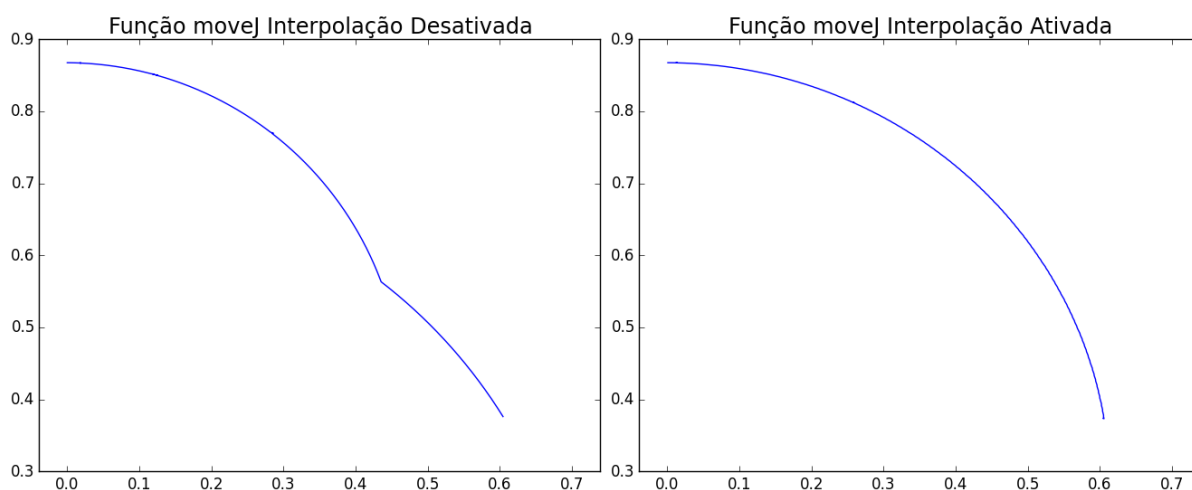
Primeiro o robô é posicionado no conjunto de ângulos  $(90^\circ, 0^\circ)$ , e em seguida realiza movimento para o conjunto  $(0^\circ, 70^\circ)$ . É possível observar a trajetória descrita pela extremidade do manipulador na Figura 39 abaixo.



**Figura 39:** Comparação entre funções com e sem interpolação de juntas.

A interpolação de juntas, nada mais é do que o cálculo realizado pelo manipulador, para desenvolver o movimento das juntas de modo que ambas terminem seu movimento juntas, para isso, é necessário adaptar a velocidade de movimentação de cada junta.

Nota-se, porém, que a posição inicial e final do manipulador permanece as mesmas, o que é alterado é apenas a trajetória desenvolvida pelo manipulador, conforme observado na Figura 40 abaixo.



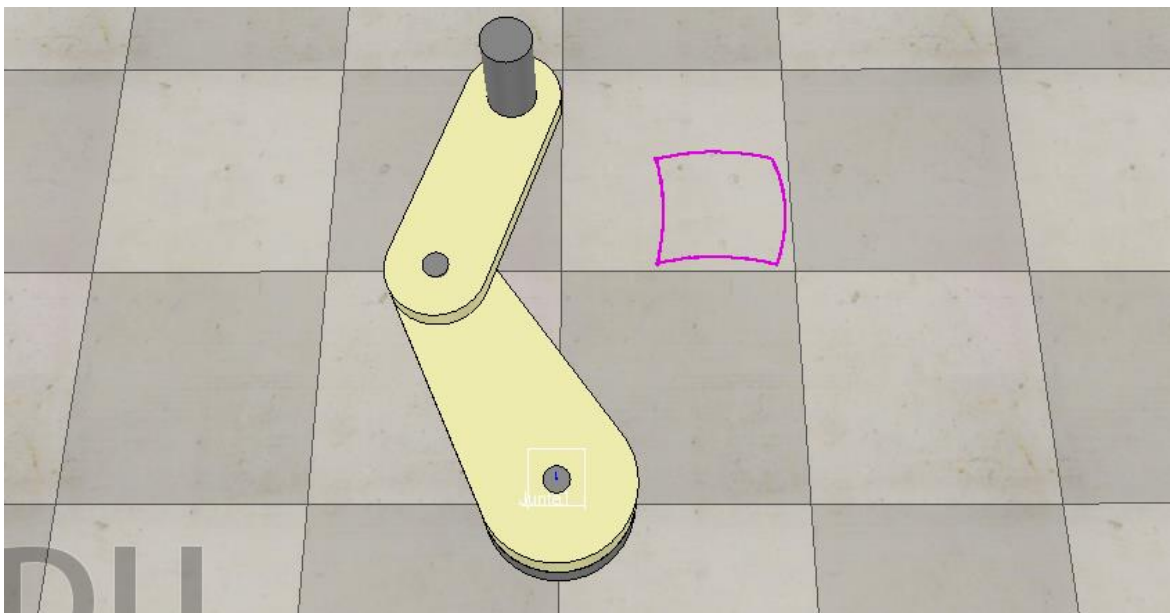
**Figura 40:** Gráficos das funções com interpolação ligada e desligada.

## 4.2. MOVIMENTO ENTRE PONTOS

Para testar a função *moveP*, foi desenvolvido a rotina de programação para posicionar a extremidade do manipulador em cada um dos pontos de um quadrado, utilizando o conceito de movimentação por *offset*, assim, foi especificado um ponto chamado *home*, e a partir desse ponto foi aplicado um *offset* com o título de *side*, que representa o tamanho dos lados do quadrado.

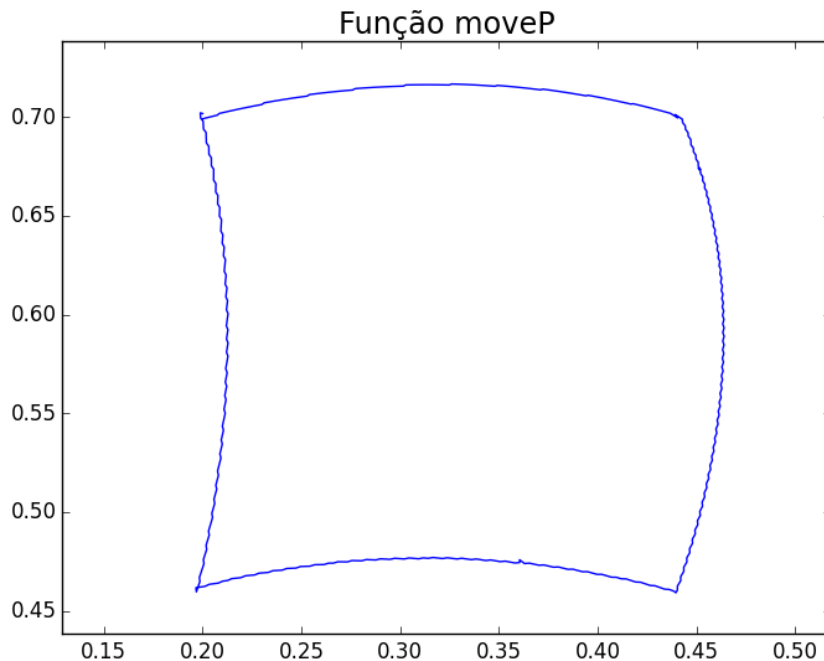
```
1. void teste_movep()  
2. {  
3.     //Define tamanho de lado do quadrado  
4.     float side = 0.24f;  
5.     //Define ponto de referência para o quadrado  
6.     vector_position home = create_point(0.20f, 0.7f);  
7.     //Movimenta robô para ponto de referência  
8.     moveP_Offs(home, 0, 0, true);  
9.     plot_start();  
10.    //Realiza movimento pelos pontos de um quadrado  
11.    moveP_Offs(home, side, 0, true);  
12.    moveP_Offs(home, side, -side, true);  
13.    moveP_Offs(home, 0, -side, true);  
14.    moveP_Offs(home, 0, 0, true);  
15.    plot_end();  
16. }
```

Na Figura 41 abaixo, é possível observar a trajetória descrita pelo manipulador no teste realizado. As quatro coordenadas inseridas como alvo de movimentação são os quatro vértices do quadrado.



**Figura 41:** Resultado da rotina de simulação com função de movimento entre pontos.

Observa-se, porém, que não estão sendo desenhadas as linhas do quadrado, isso fica a cargo da função *moveL*, este primeiro teste tem como objetivo apenas posicionar a extremidade pelos vértices do quadrado. Deste modo, a trajetória entre os pontos é descrita como um arco entre os pontos, característica de um movimento com interpolação de juntas, conforme observado na Figura 42 abaixo.



**Figura 42:** Gráfico de resultado da rotina com função de movimento entre pontos.

#### 4.3. MOVIMENTO LINEAR

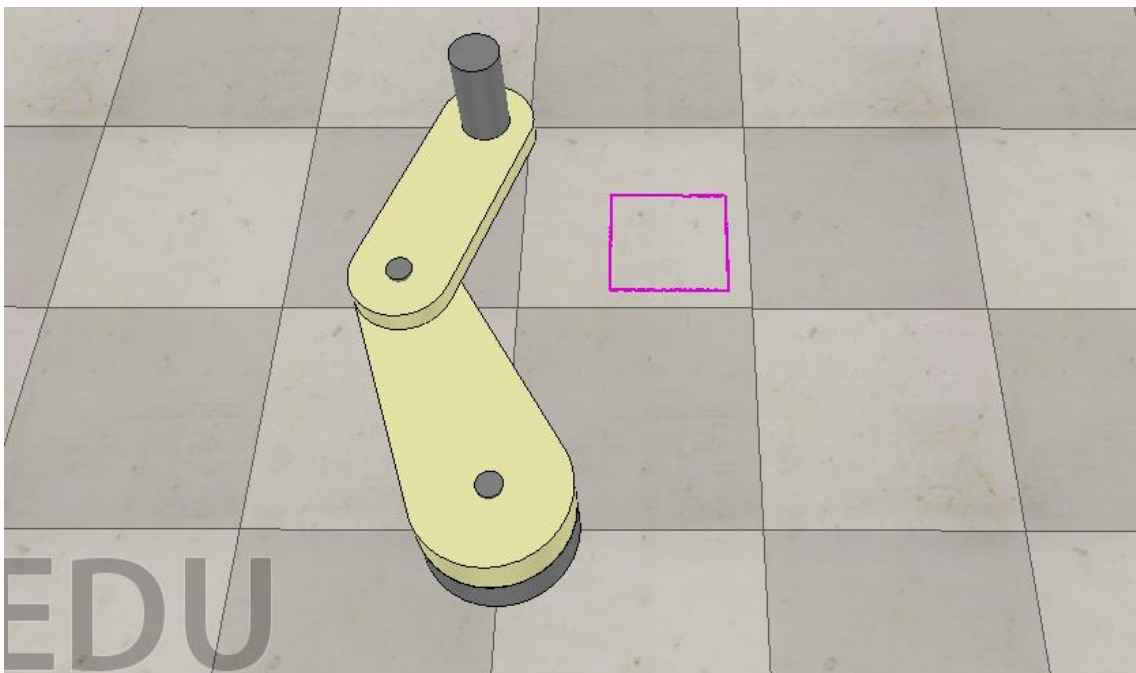
O funcionamento da função *moveL* desenvolvida, é semelhante a *moveP*, realizando movimentação entre dois pontos no plano cartesiano, entretanto, dessa vez a trajetória descrita é considerada, onde o manipulador descreve uma linha reta entre os pontos especificados.

Para o teste da função *moveL*, inicialmente foram considerados os mesmos pontos do quadrado do teste da função *moveP*, realizando, porém, o movimento com as funções de interpolação linear, para descrever a trajetória das arestas do quadrado, não somente dos vértices.

O trecho de código abaixo apresenta a rotina utilizada para este teste, utilizando novamente o conceito das funções com *offset*, onde é inserido um ponto de referência para a figura, e aplicado valores de *offset* a partir da referência.

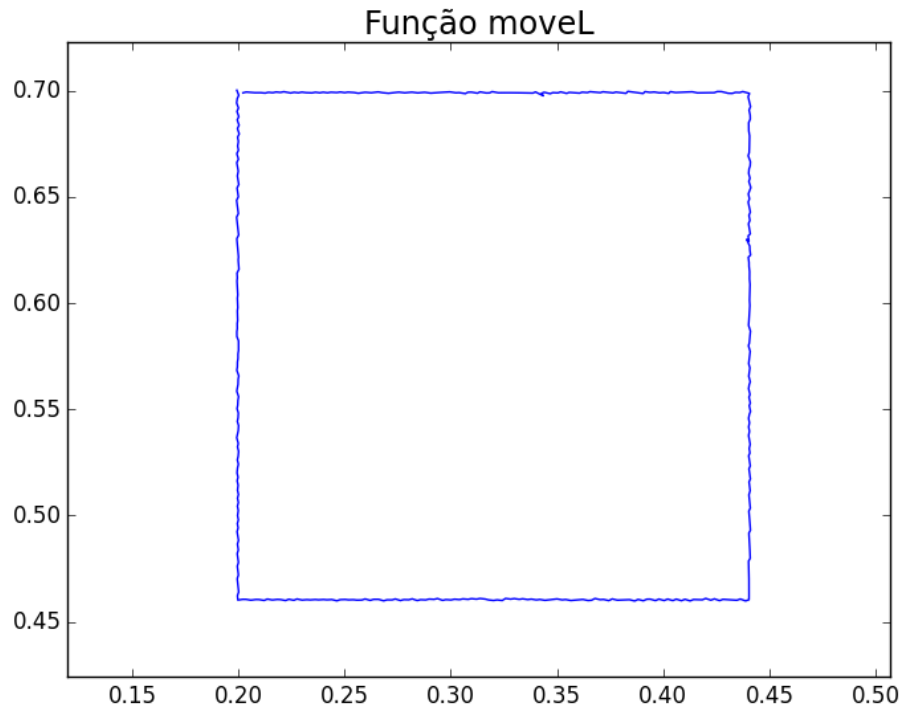
```
1. void teste_movel()
2. {
3.     //Define tamanho de lado do quadrado
4.     float side = 0.24f;
5.     //Define quantidade de pontos da reta
6.     int points = 120;
7.     //Define ponto de referência para o quadrado
8.     vector_position home = create_point(0.20f, 0.7f);
9.     //Movimenta robô para ponto de referência
10.    moveP_Offs(home, 0, 0, true);
11.    //Realiza movimento pelos pontos de um quadrado
12.    plot_start();
13.    moveL_Offs(home, side, 0, points);
14.    moveL_Offs(home, side, -side, points);
15.    moveL_Offs(home, 0, -side, points);
16.    moveL_Offs(home, 0, 0, points);
17.    plot_end();
18. }
```

Para a função de movimento linear, o terceiro parâmetro é especificado como um valor inteiro que indica a quantidade de pontos que a reta descrita contém, valores mais altos geram retas mais precisas. A Figura 43 abaixo apresenta a trajetória do robô SCARA executando o código apresentado.



**Figura 43:** Resultado da rotina de simulação com movimentação linear.

É possível notar a diferença entre as características do movimento com a função *moveL* para a função *moveP*, embora sejam realizadas a partir dos mesmos pontos. A Figura 44 a seguir apresenta o resultado do gráfico gerado com os pontos da trajetória do teste com a função *moveL*.

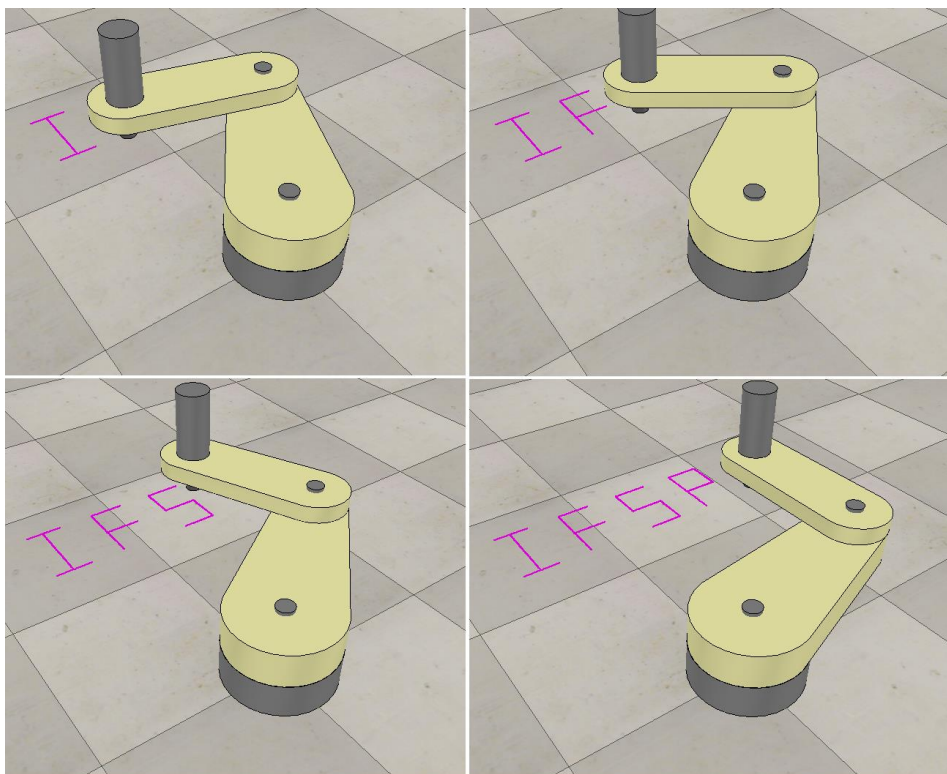


**Figura 44:** Gráfico de resultado da simulação com movimento linear.

Ainda com a função *moveL*, foram realizados outros testes de escrita, como por exemplo da palavra “IFSP.” A base para o código é basicamente a mesma que do teste anterior, uma série de combinações de função de interpolação linear juntas, especificando os pontos desejados e o momento certo de coletar as coordenadas da extremidade do braço, como se houvesse uma caneta como ferramenta no manipulador.

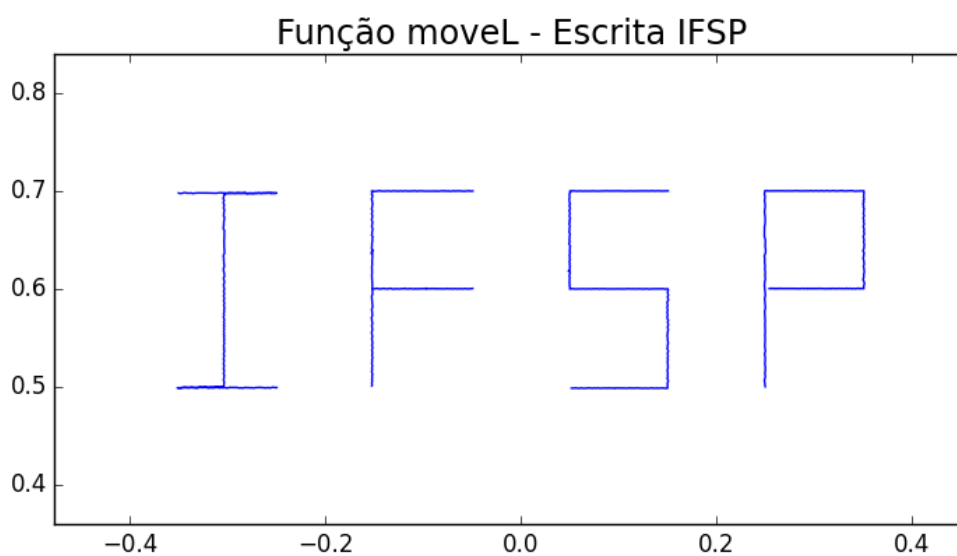


A Figura 45 a seguir apresenta o resultado da rotina de programação responsável pela escrita da palavra “IFSP” no *software* V-Rep.



**Figura 45:** Teste de simulação de escrita com função linear.

Já na Figura 46 a seguir, é possível observar a trajetória dos pontos coletados descrita em forma de gráfico com a biblioteca *Matplotlib*.



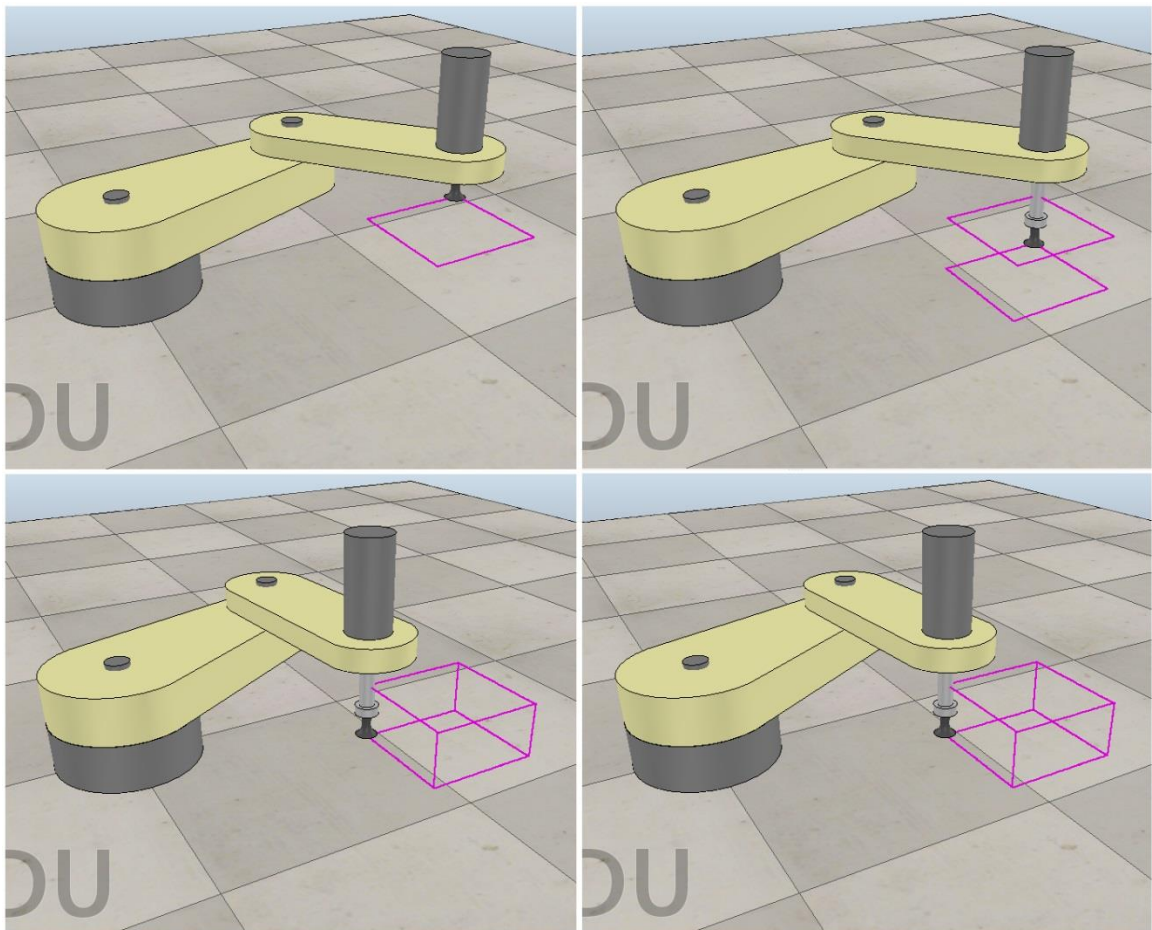
**Figura 46:** Gráfico de resultado da simulação de escrita com função linear.

#### 4.4. MOVIMENTO NO EIXO Z

Os testes apresentados consideraram apenas o plano cartesiano como espaço de trabalho, ou seja, todas as figuras e trajetórias foram traçadas em um plano bidimensional. Para o teste de um plano tridimensional, foi programada a rotina para desenvolver a trajetória pelos pontos de um paralelepípedo.

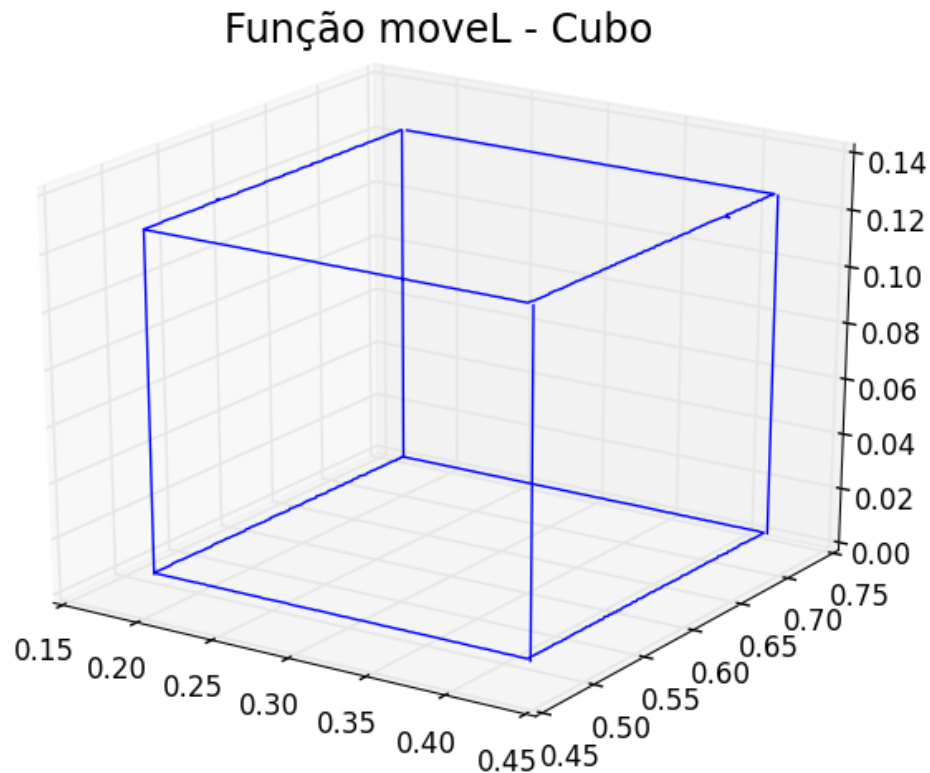
Para desenvolver uma trajetória tridimensional, foi utilizada a função *moveZ*, realizando movimento da junta prismática do manipulador, localizando a extremidade do SCARA em um plano tridimensional, e de igual modo, os pontos da extremidade foram coletados para a construção dos gráficos.

A Figura 47 a seguir, apresenta a trajetória desenvolvida no *software* V-Rep, desenvolvendo uma figura tridimensional semelhante a um cubo, com formato paralelepipedal.



**Figura 47:** Resultado da simulação em um plano tridimensional.

Semelhanamente aos outros testes foi construído um gráfico com os pontos coletados e exportados pelo *software* de simulação, visualizado na Figura 48 abaixo. Nele é possível perceber novamente a influência da quantidade de pontos especificada para a interpolação linear com a precisão da reta.



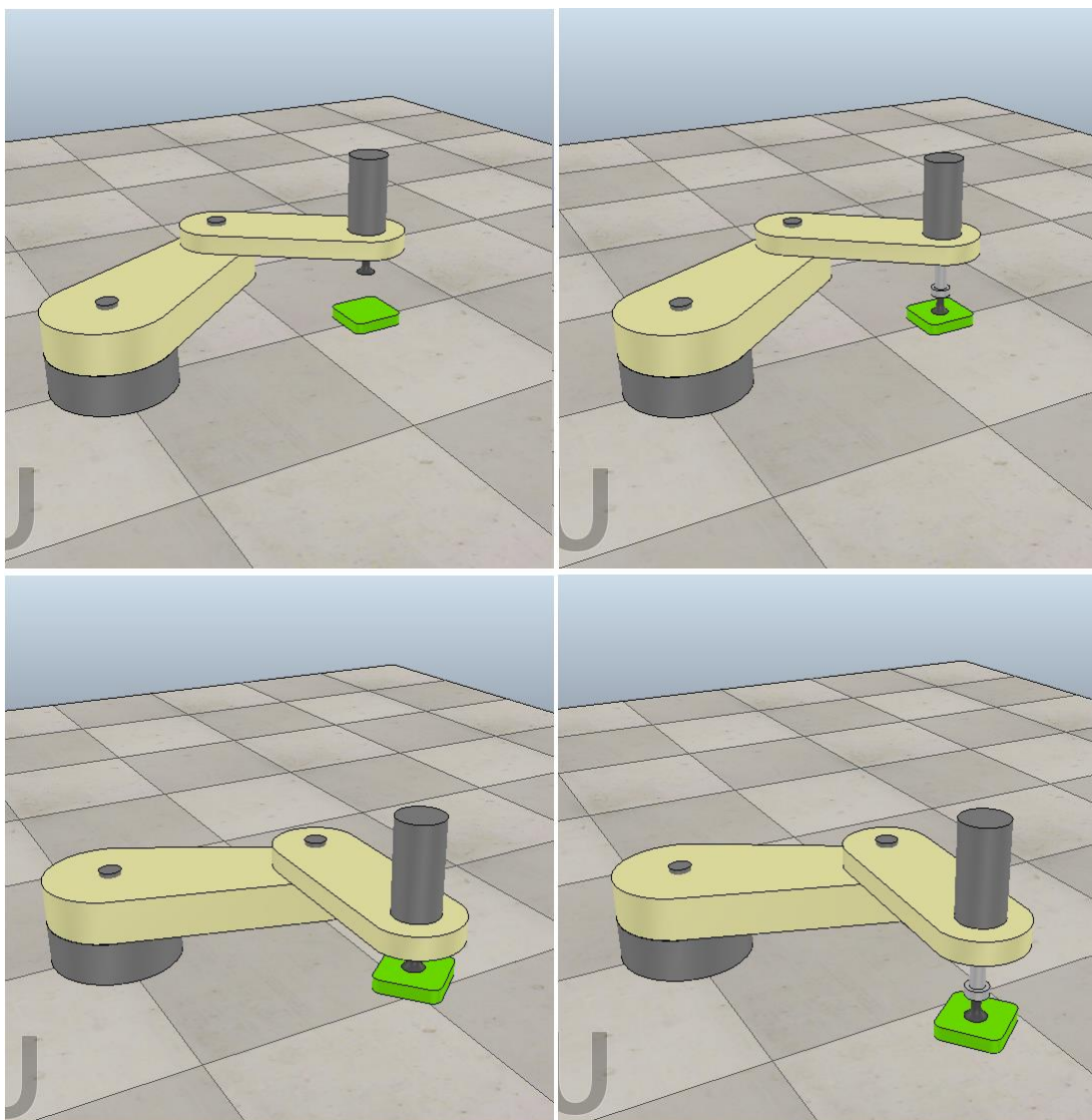
**Figura 48:** Gráfico de resultado da simulação em um plano tridimensional.

É possível notar algumas falhas de precisão na trajetória desenvolvida, onde alguns vértices da figura não se encontram perfeitamente, isso se deve a quantidade de pontos especificada para as trajetórias das retas, assim como a precisão escolhida para a movimentação das juntas.

#### 4.5. CAPTURA E POSICIONAMENTO DE OBJETOS

Para as rotinas de captura e posicionamento de peça, foram utilizadas as funções desenvolvidas para controle da ferramenta de sucção citadas anteriormente, posicionando a extremidade do robô sob uma peça, movimentando o eixo z e ativando a ferramenta de sucção no momento certo.

Neste teste, foi utilizada a propriedade do *software* de simulação V-Rep de interação com outros objetos sólidos, o que permite que a ferramenta de sucção “capture” peças e possa carregá-las junto com o movimento do manipulador. A Figura 49 abaixo mostra o passo a passo de um exemplo de captura e posicionamento de peça.

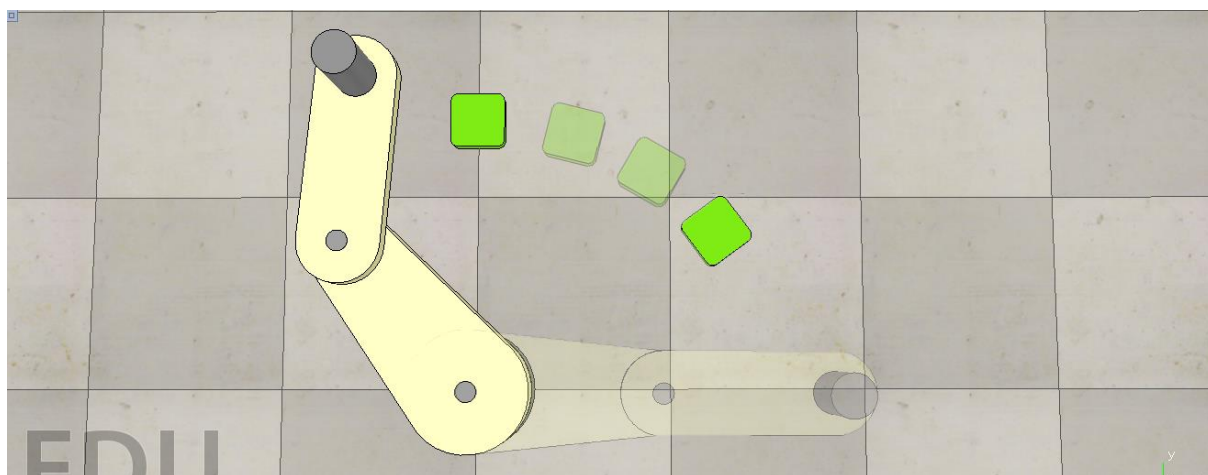


**Figura 49:** Teste de captura e posicionamento de objetos.



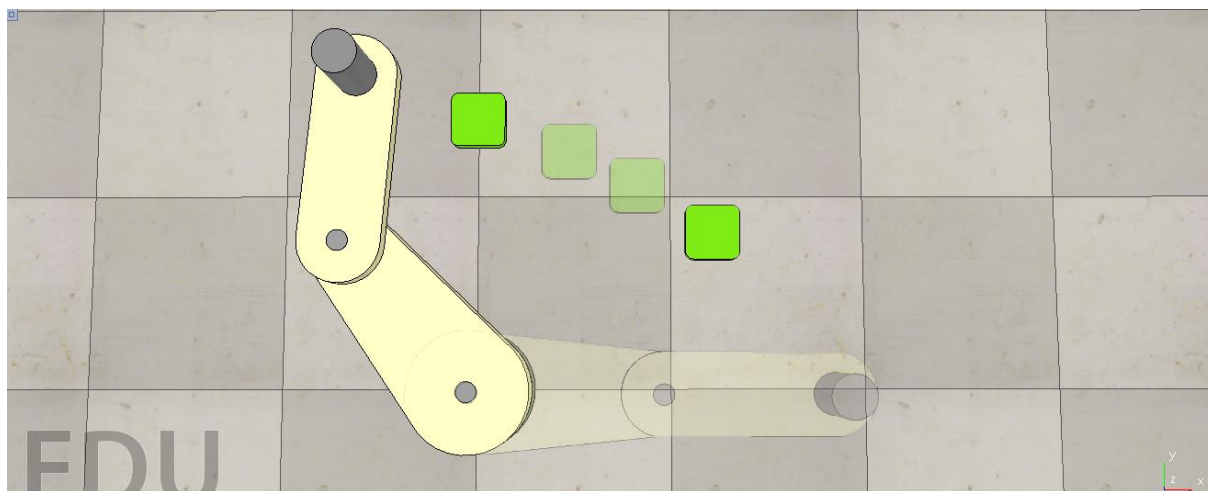
Durante a captura e posicionamento de objetos, deve ser observado o comportamento da extremidade do manipulador quanto à velocidade angular em função da adaptação de juntas rotativas para um movimento em plano cartesiano.

A extremidade do robô ao capturar uma peça, pode mantê-la fixa, ou utilizar da junta rotacional na ferramenta do manipulador para rotacionar a peça juntamente com o movimento realizado. A Figura 50 abaixo demonstra um movimento de captura e posicionamento sem ação de correção da velocidade angular.



**Figura 50:** Captura e posicionamento sem ação de correção de velocidade angular.

É possível notar que a peça ao ser capturada, acompanha a rotação da extremidade do braço, que permanece fixa, gerando assim ao final do movimento, uma distorção na posição da peça quanto a sua posição inicial. Já a Figura 51 a seguir, apresenta um movimento com a ação da correção de velocidade angular.

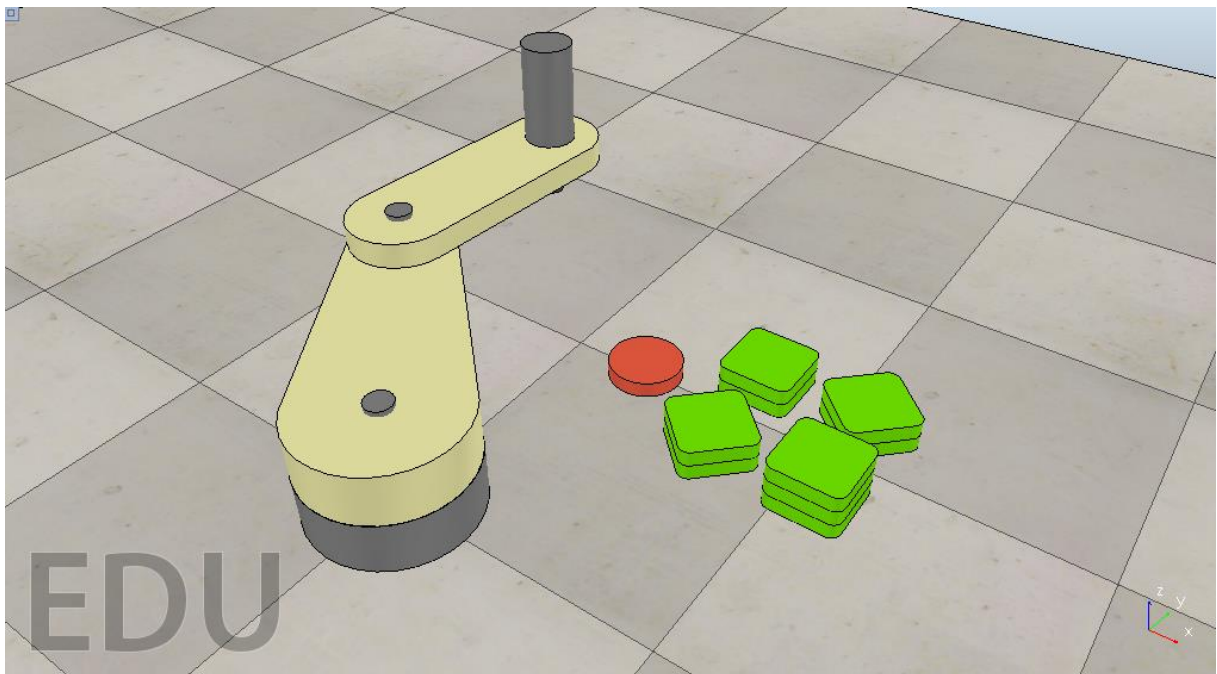


**Figura 51:** Captura e posicionamento com ação de correção de velocidade angular.

#### 4.6. TESTE DE MONTAGEM

Por fim, foi realizado um teste de montagem com necessidade de correção de velocidade angular e captura e posicionamento de peças, para colocar todas as funções desenvolvidas até então em funcionamento juntas, o objetivo foi simular uma situação de montagem.

Simulando um ambiente de montagem real, a intenção foi levar em consideração a orientação e posicionamento das peças para formar uma montagem final. As peças foram modeladas e exportadas utilizando o *software Solid Works* e importadas para o V-Rep com as ferramentas nativas do programa.

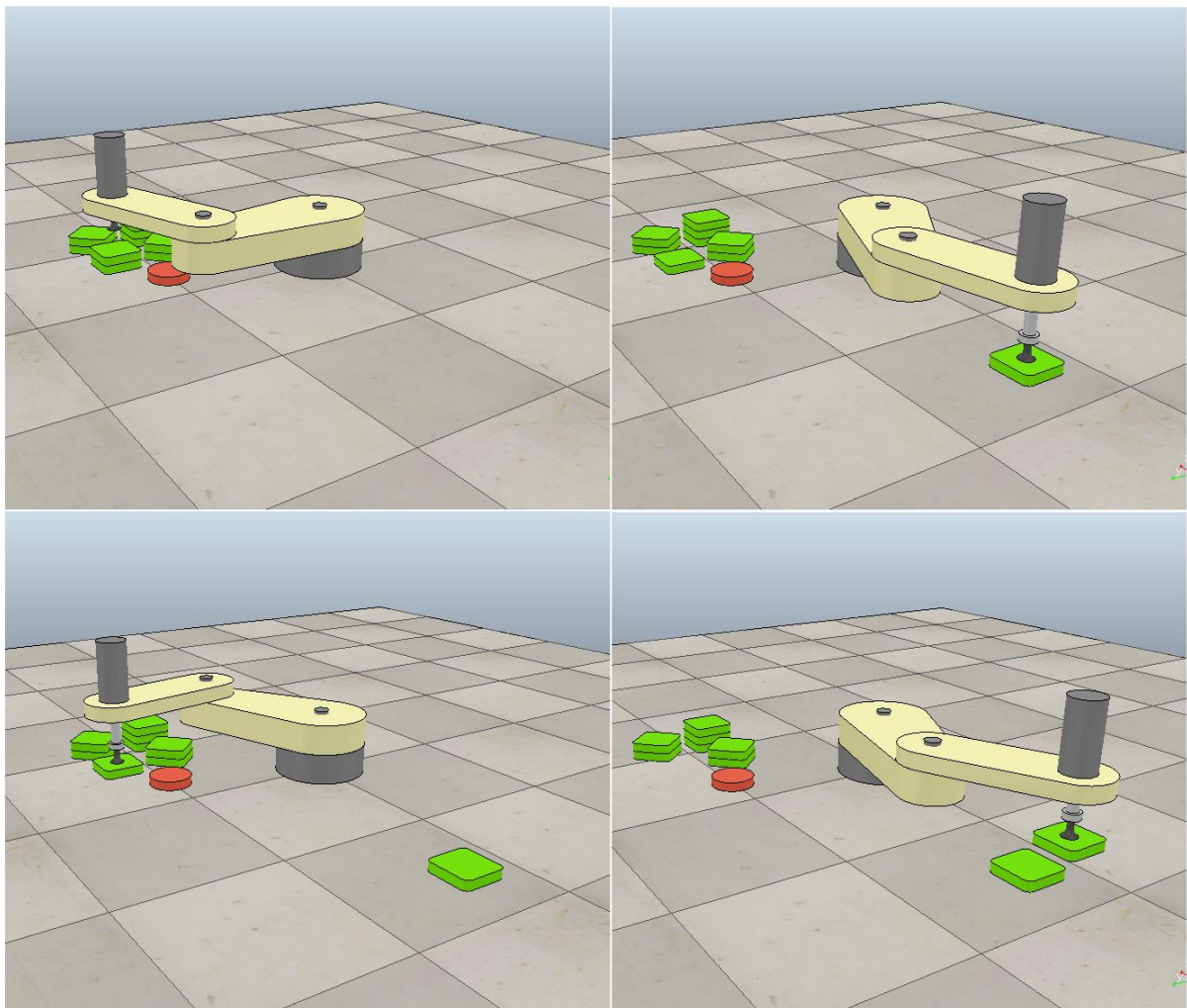


**Figura 52:** Ambiente montado para simulação de posicionamento.

Inicialmente as peças foram dispostas em posições e orientações diferentes, sem formar nenhuma configuração que fizesse sentido, ao final serão organizadas de modo a formar o logo do Instituto Federal de Ciência e Tecnologia de São Paulo (IFSP).

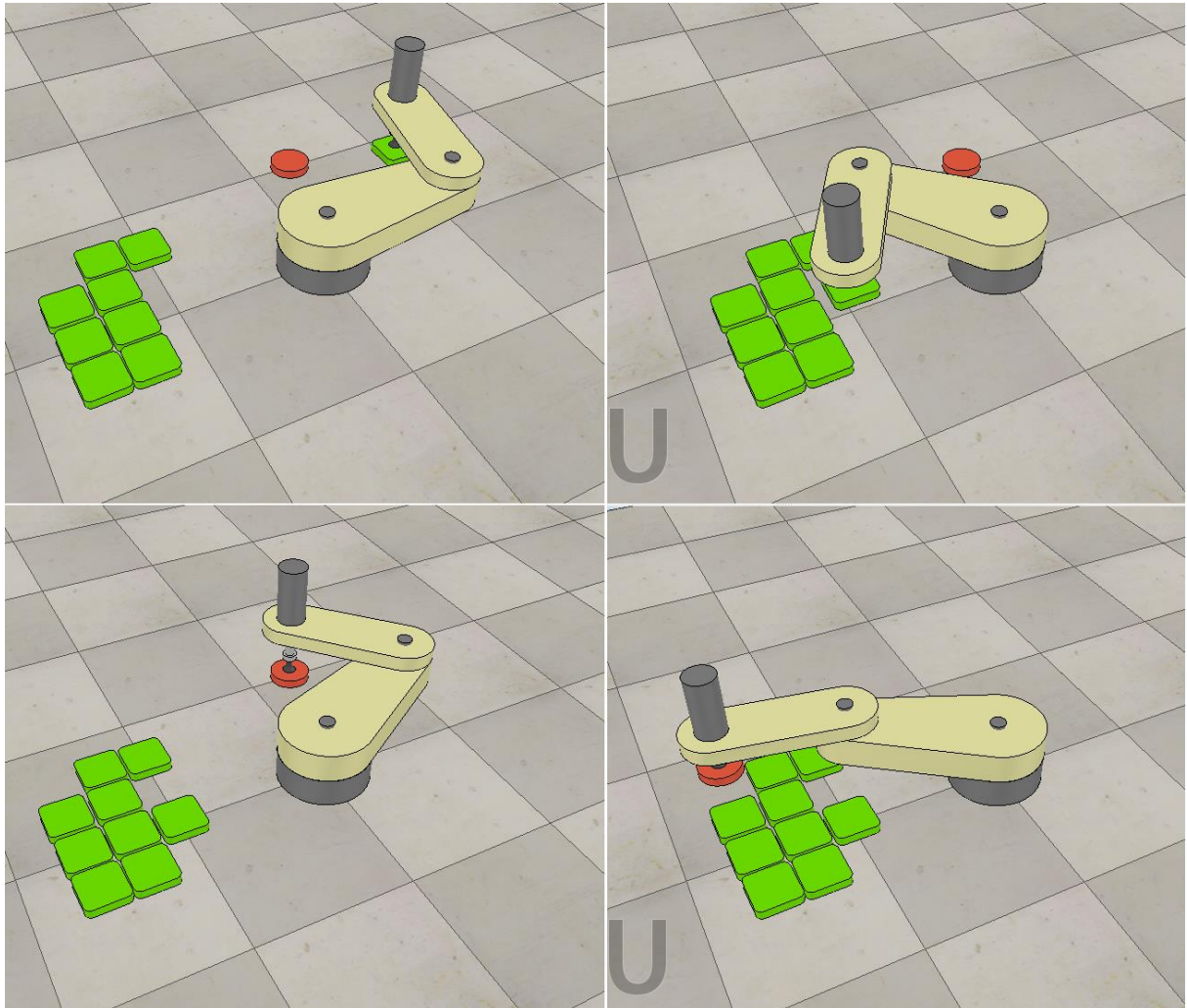
As peças foram sendo capturadas uma a uma e levadas a sua posição final com as funções *moveJ* e *moveP*. Quando necessário, a movimentação da junta rotacional reorientava a peça conforme a função *moveR*, além disso, as peças se encontravam empilhadas em alturas diferentes, sendo necessário o correto uso da função *moveZ* para o controle do eixo z.

A todo instante, a correção angular se fez ativa, levando em consideração a importância da orientação correta das peças para o encaixe final. A Figura 53 abaixo mostra brevemente algumas etapas da rotina sendo executada.



**Figura 53:** Simulação de rotina com processo de montagem.

A rotina programada continua até executar a captura e posicionamento de todas as peças dispostas inicialmente ao lado do manipulador. A Figura 54 abaixo mostra as ultimas peças sendo posicionadas.

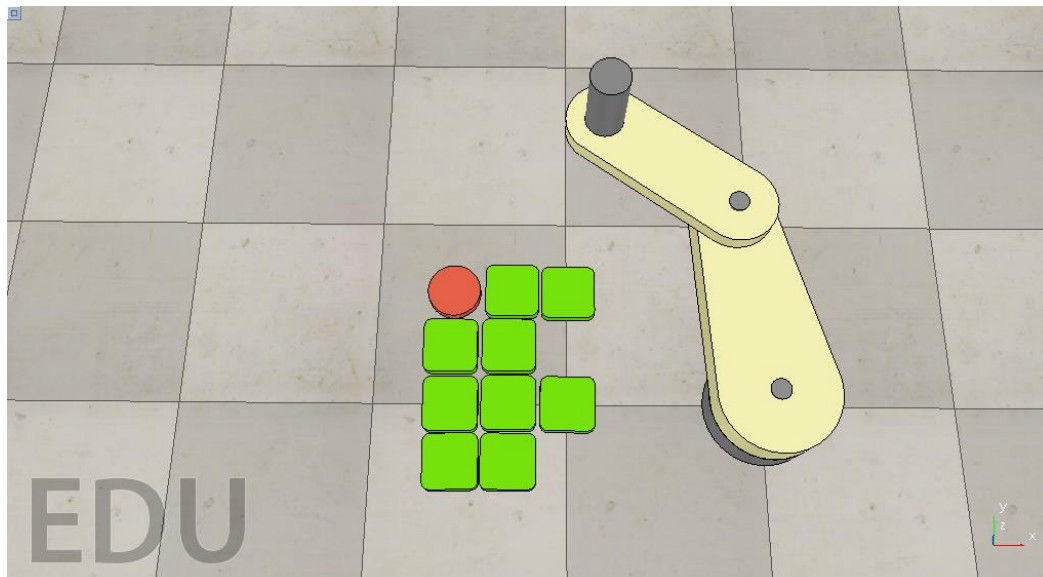


**Figura 54:** Últimas etapas da rotina de simulação de montagem.

O manipulador durante o desenvolvimento da montagem, não posicionou nenhuma peça sob a outra, mostrando que o modelo matemático desenvolvido e programado se encontra preciso o suficiente para realizar trajetórias dentro do *workspace* proposto.

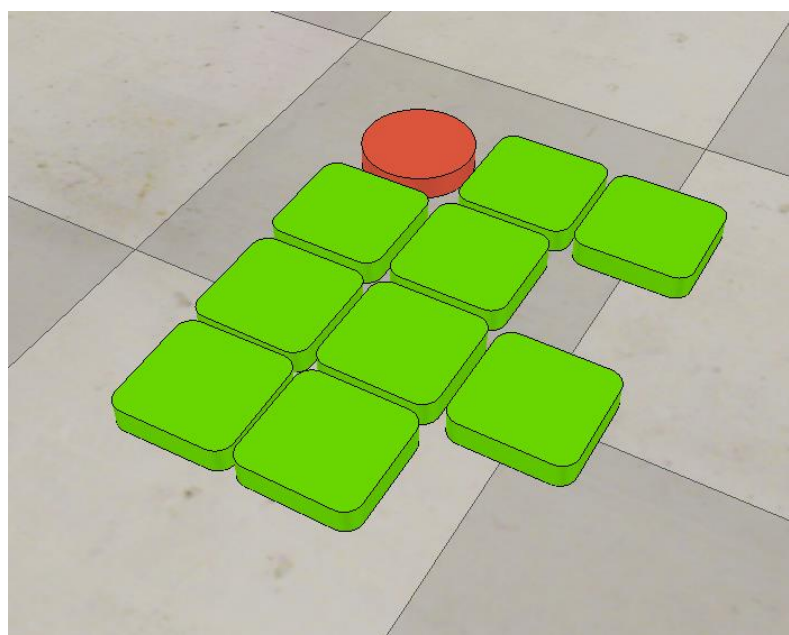


A montagem finalizada pode ser observada na Figura 55 a seguir. É possível notar que a montagem foi executada com precisão aceitável no posicionamento e orientação das peças.



**Figura 55:** Simulação de montagem finalizada.

A Figura 56 abaixo mostra a montagem final com um nível maior de aproximação, permitindo a observação com maior precisão a montagem realizada pela rotina programada.



**Figura 56:** Montagem finalizada com maior nível de detalhes.

## 5. CONCLUSÃO

O trabalho proposto tinha como objetivo realizar a modelagem matemática e simulação de um robô manipulador do tipo SCARA, utilizando técnica *Hardware in the Loop*, integrando as etapas de desenvolvimento matemático, programação em sistema embarcado e simulação em *software* supervisor.

A etapa inicial, ao que refere ao modelo matemático, foi desenvolvida como o esperado, gerando equações através do sistema *Denavit-Hartenberg* e modelo trigonométrico que descreveram corretamente o comportamento do mecanismo robótico proposto. O modelo desenvolvido foi testado e teve sua funcionalidade averiguada através da programação em linguagem Python.

Durante a execução do modelo matemático, foram propostas duas abordagens diferentes para descrever a localização do mecanismo robótico no espaço, sendo elas a cinemática inversa e cinemática direta. Através de prova por *software* com o modelo desenvolvido e programado, foi possível validar o funcionamento de cada uma delas, realizando a transferência do espaço cartesiano para o espaço das juntas e também o contrário.

Quanto ao *software* de simulação, o V-Rep se mostrou funcional e suficiente para realizar a atividade proposta, apesar de ter sido utilizado apenas os recursos necessários para a simulação do manipulador SCARA. O projeto proporcionou melhor entendimento do funcionamento do *software* e validou a utilização da simulação como mecanismo de integração com plataforma embarcada e *Hardware in the Loop*.

Com o mesmo *software*, seria possível ainda simular diferentes ambientes, envolvendo outras ferramentas, como garras, atuadores de diferentes tipos, sistemas de visão, entre outros. No entanto a proposta inicial do projeto era utilizar a simulação como representação visual dos cálculos e rotinas executadas em *hardware*, o que foi contemplado pela metodologia desenvolvida com êxito.

A integração com o sistema HIL no projeto desenvolvido, foi executada com um grau de funcionalidade satisfatória, cumprindo os objetivos iniciais de aplicar o modelo matemático em plataforma embarcada e realizar rotinas de movimentação com o manipulador tridimensional. As etapas anteriores, abstratas quando observadas individualmente, foram perfeitamente visualizadas em conjunto quando integradas com o sistema de simulação embarcada.

Como propostas futuras, é possível inserir diversas funcionalidades no projeto desenvolvido, como por exemplo, novas ferramentas, sensores, integração com outros robôs ou linha de produção, bem como aprimoramento do modelo desenvolvido para obter ganhos com precisão e eficácia. Além disso, é possível também desenvolver novas funções de movimento presentes em linguagens reais como trajetórias de arco e círculos.

Além das melhorias em *software* e modelo tridimensional, é possível também propor a montagem real de um manipulador SCARA, uma vez que o código necessário para seu controle foi desenvolvido e testado por meio das simulações executadas.

Por fim, o intuito do trabalho apresentado além da construção teórica e prática, foi também disseminar a atenção educacional para a utilização da tecnologia *Hardware in the Loop*, juntamente com o *software* de simulação V-Rep como ferramenta de supervisão e aprendizado ao que tange o campo da robótica. Espera-se que o conteúdo aqui apresentado venha a ser útil em projetos ou consultas futuras em trabalhos e atividades nas áreas de conhecimento agregado.

## REFERÊNCIAS BIBLIOGRÁFICAS

ARDUINO. **Arduino**. 2018. Disponível em: <<https://www.arduino.cc/>>. Acesso em: 4 set. 2018.

ASSOCIATION, Robotics Industries. **Defining The Industrial Robot Industry and All It Entails**. 2017. Disponível em: <<https://www.robotics.org/robotics/industrial-robot-industry-and-all-it-entails>>. Acesso em: 15 nov. 2018.

BECKER, Marcelo. **Cinemática Direta de Manipuladores Robóticos**. São Paulo: Eesc-usp, 2008. 64 slides, color.

CRAIG, John. **Robótica**. 3. ed. São Paulo: Pearson, 2013. 392 p.

GEOGEBRA. **GeoGebra: Aplicativos Matemáticos**. 2018. Disponível em: <<https://www.geogebra.org/>>. Acesso em: 20 set. 2018.

INSTRUMENTS, National. **Arquiteturas de sistemas de teste hardware-in-the-loop (HIL)**. 2016. Disponível em: <<http://www.ni.com/white-paper/10343/pt/>>. Acesso em: 23 ago. 2018.

LIU, Junyi et al. Four Degrees of Freedom SCARA Robot Modeling and Simulation. **2014 International Symposium On Computer, Consumer And Control**, [s.l.], p.410-412, jun. 2014. IEEE. <http://dx.doi.org/10.1109/is3c.2014.113>.

NIKU, Saeed B.. **Introdução à Robótica: Análise, Controle, Aplicações**. 3. ed. [s. L.]: Ltc, 2013. 404 p.

ORG, Python. **Python**. 2018. Disponível em: <<https://www.python.org/>>. Acesso em: 22 set. 2018.

PAZOS, Fernando. **Automação de Sistemas e Robótica**. São Paulo: Axcel Books, 2002. 392 p.

PIYARATNA, Sanka et al. Digital RF processing system for Hardware-in-the-Loop simulation. **2013 International Conference On Radar**, [s.l.], p.554-559, set. 2013. IEEE. <http://dx.doi.org/10.1109/radar.2013.6652048>.

RIVIN, Eugene I.. **Mechanical Design of Robots**. New York: Mcgraw Hill Higher Education, 1987. 368 p.

ROBOTICS, Coppelia. **V-Rep**: Virtual Robot Experimentation Platform. 2015. Disponível em: <<http://www.coppeliarobotics.com/>>. Acesso em: 8 ago. 2018.

ROSÁRIO, João Maurício. **Princípios de Mecatrônica**. São Paulo: Pearson Brasil, 2005. 368 p.

SARHADI, Pouria; YOUSEFPOUR, Samereh. State of the art: hardware in the loop modeling and simulation with its applications in design, development and implementation of system and control software. **International Journal Of Dynamics And Control**, [s.l.], v. 3, n. 4, p.470-479, 10 jun. 2014. Springer Nature. <http://dx.doi.org/10.1007/s40435-014-0108-3>.